UM ESQUEMA DE RECONFIGURAÇAO HEURISTICO PARA UM COMPUTADOR PARALELO TOLERANTE A FALHAS

S. W. Song
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
C.P. 20570, CEP 01498 São Paulo, SP

RESUMO

Este artigo descreve uma arquitetura altamente reconfigurável para um arranjo retangular bi-dimensional de processadores poderosos. Devido ao alto grau de reconfigurabilidade, a arquitetura pode prover tolerância a falhas com eficiente utilização dos processadores e servir de suporte a programas de aplicação que requerem diferentes estruturas de interconexão. O esquema de reconfiguração considera o arranjo físico como um grafo qualquer, podendo ser aplicado não apenas a arranjos retangulares. Em termos de considerações de falhas, supomos não apenas falhas de processadores, mas também as de chaves e canais físicos de comunicação. Nesses dois aspectos o presente trabalho distingue-se dos anteriores.

ABSTRACT

This paper describes a highly reconfigurable architecture for two-dimensional mesh-connected arrays of powerful processors. Because of its high degree of reconfigurability the architecture can provide fault tolerance with efficient array utilization and support application programs requiring different interconnection structures. The reconfiguration scheme considers the physical array as a graph, and can thus be applied not only to mesh-connected arrays. In terms of failure considerations, we assume not only processor failures, but also failures of switches and physical communication channels. In these two respects the current work is distinct from the previous ones.

Este trabalho contou com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob a bolsa de pesquisa - processo nº 306063.

1. INTRODUÇAO

Vários arranjos de processadores bi-dimensionais reconfiguráveis têm sido propostos na literatura [1, 2, 3]. O arranjo apresentado neste artigo difere-se da maioria dos já propostos em dois aspectos. Em primeiro lugar, ele é um arranjo de processadores poderosos, em contraste aos elementos de processamento simples usualmente usados. Em segundo lugar, além de prover tolerância a falhas, ele é altamente reconfigurável para dar apoio a programas de aplicação que requerem diferentes estruturas de interconexão.

Tradicionalmente arranjos de processadores bi-dimensionais têm sido considerados para um número grande de elementos de processamento, com capacidade limitada de programabilidade e processamento. Pesquisadores na Universidade Carnegie Mellon desenvolveram um arranjo de processadores uni-dimensional chamado Warp (atualmente sendo manufaturado pela G.E.) que possui processadores poderosos (com 10 MFLOPS cada) e é dirigido para um amplo domínio de aplicações [4]. Como um esforço

subsequente, foi feita uma investigação em conjunto com a G.E. sobre um arranjo reconfigurável bi-dimensional usando os mesmos processadores. Neste artigo, apresentamos resultados daquele estudo, dando ênfase ao esquema de reconfiguração, que foi desenvolvido pelo autor.

Estamos interessados em reconfigurabilidade por duas razões: para prover tolerância a falhas e para aumentar a programabilidade. Um arranjo reconfigurável com processadores redundantes pode tolerar um número de processadores defeituosos e continuar implementando a estrutura requerida pelo programa. Portanto, arranjos reconfiguráveis são muitas vezes considerados para arranjos de processadores para implementação VLSI/WSI a fim de melhorar o *yield* na fabricação [5, 6]. Estamos também interessados em melhorar a sobrevivência do arranjo para aplicações onde reparos manuais não são viáveis e onde há um requisito mínimo na duração da vida útil do equipamento.

O grau de reconfigurabilidade determina a eficiência em utilizar processadores sem defeitos e afeta o número de processadores necessários para um dado requisito de vida útil. Portanto, reconfigurabilidade é mais crítica para arranjos de processadores complexos e poderosos do que para arranjos de processadores simples.

Além da tolerância a falhas, um alto grau de reconfigurabilidade é desejável para aumentar a programabilidade dos arranjos. Algoritmos paralelos para aplicações diferentes geralmente requerem diferentes números de processadores e diferentes interconexões. Como um amplo domínio de aplicações é desejável para justificar o custo de um arranjo de processadores poderosos, torna-se importante a possibilidade de suportar uma variedade de modelos de utilização. Isso é refletido no nosso algoritmo de reconfiguração que suporta um modelo geral tanto para o programa como para o arranjo físico de processadores.

2. ARQUITETURA

2.1. Estrutura do Arranjo

A Figura 1 mostra uma representação de um arranjo 4x4. Os quadrados representam os *processadores* que são também chamados *células*. Os retângulos na periferia da figura representam "buffers" de E/S. Os "buffers" de E/S estão também conectados a um computador hospedeiro que transfere dados ao arranjo. Os círculos representam *chaves* que são ligadas a células ou "buffers" de E/S através de linhas bi-direcionais chamadas *canais físicos*. Uma chave tem seis portas ("ports"), quatro conectadas a chaves vizinhas ou "buffers" de E/S e duas conectadas à célula local, e pode estabelecer qualquer padrão de interconexão entre suas portas.

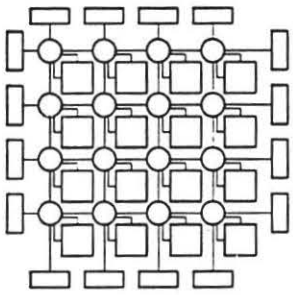

Figura 1. Um arranjo físico 4x4

A estrutura da Figura 1 pode ser usada para implementar qualquer arranjo bi-dimensional, embora a nossa ênfase aqui é nos arranjos de processadores poderosos, onde são críticas a utilização e reconfigurabilidade dos processadores. Especificamente, esta arquitetura foi desenvolvida para construir um arranjo bi-dimensional de células Warp. Faremos uma breve descrição da célula Warp para ilustrar o que queremos dizer com processador poderoso.

2.2. Célula Warp

A célula Warp é um processador de 10 MFLOPS, mostrado na Figura 2. Há duas unidades aritméticas de ponto flutuante de 32 bits, um multiplicador e um somador, cada qual com seus próprios registros de 32 palavras para armazenamento dos operandos. Uma memória local de 32K palavras e uma memória de "overflow" de 2K palavras

são usadas para guardar dados residentes e temporários, podendo ser endereçadas ou por um microcódigo literal ou um endereço computado. A célula se comunica com outras através de duas vias de E/S chamadas X e Y. Cada via pode transferir até um total de 40 Mbytes por segundo com as células vizinhas e possui uma fila de entrada de 512 palavras para armazenar dados de entrada. Todas as componentes são ligadas por uma chave livre de bloqueio do tipo "cross-bar", com uma largura de banda interna de 120 Mbytes por segundo. Unidades funcionais múltiplas, altas larguras de banda interna e externa e memória local grande são combinadas para tornarem a célula Warp um potente engenho de computação e distinguem-na dos muitos outros processadores usados em arranjos sistólicos.

Cada célula Warp é equipada com sua própria memória de programa de 8K instruções e micro-sequenciador, com instruções largas de mais de 250 bits. Como cada célula pode ser programada independentemente, um modelo de programação heterogêneo pode ser suportado. As células são programadas numa linguagem de alto nível chamada W2 e código é gerado por um compilador otimizador [7]. A célula Warp é implementada numa placa de 15" x 17" com mais de 270 pastilhas.

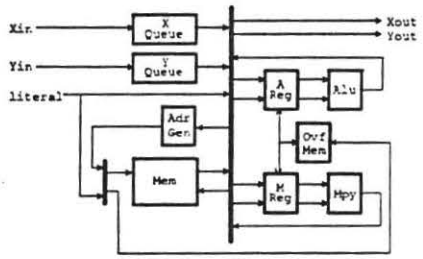

Figura 2. Uma célula Warp

2.3. Reconfigurabilidade e Tolerância a Falhas

Tolerância a falhas pode ser provida se componentes defeituosas podem ser detectadas e isoladas por reconfiguração. Testabilidade e métodos para testar arranjos de processadores são apresentados em [8] e [9]. Neste artigo, focalizamos na reconfigurabilidade e não consideramos a deteção de falhas. Supomos apenas que componentes defeituosas podem ser detectadas de algum modo.

Consideramos um modelo geral de programas onde a conectividade requerida pelo programa é representada por um grafo lógico de células e "buffers" de E/S. A reconfiguração requer o mapeamento da estrutura lógica do programa na estrutura física do arranjo que se altera à medida que falhas ocorrem ao longo do tempo, como veremos na Seção 3 a seguir.

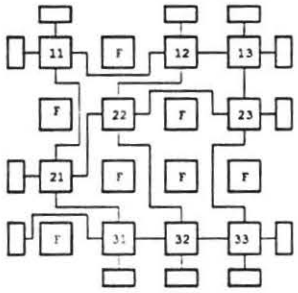

Figura 3. Um arranjo 4x4 reconfigurado para implementar um arranjo 3x3 - F denota célula com defeito

O arranjo da Figura 1 teria reconfigurabilidade limitada se cada canal físico fosse usado para implementar apenas uma conexão lógica. A medida que componentes do arranjo falham, mapeamento das conexões lógicas do programa sobre o arranjo físico iria precisar de múltiplos canais entre chaves vizinhas. Isso pode ser constatado no mapeamento do arranjo lógica 3x3 sobre o arranjo físico 4x4 com células defeituosas da Figura 3. Prover tantos canais físicos quantos são necessários, entretanto, constitui uma alternativa de desperdício e não realística, principalmente para arranjos grandes. Uma outra alternativa é multiplixar o uso de um canal físico entre múltiplas conexões lógicas. Essa solução é atraente porque ela possibilita alta reconfigurabilidade com degradação graciosa no desempenho do arranjo devida a possível congestão de canais físicos.

2.4. Interconexão Usando Canais Virtuais

Consideramos que um canal físico na Figura 1 pode implementar um número de *canais virtuais* em qualquer sentido. Por exemplo, a Figura 4 mostra o arranjo virtual visto pelo algoritmo

de reconfiguração, onde o número de canais virtuais implementados por um canal físico é quatro, dois em cada sentido. No nosso projeto atual da chave, consideramos 16 canais virtuais por canal físico.

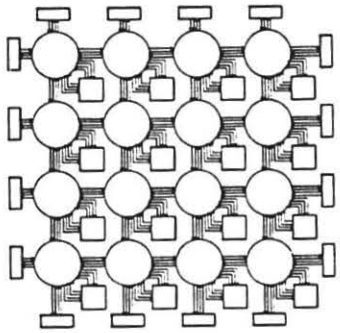

Figura 4. Canais virtuais implementados pelo arranjo físico da Figura 1

Cada conexão lógica rquerida pelo programa é mapeada num caminho dedicado virtual que começa e termina em células ou "buffers" de E/S, e consiste em uma cadeia de canais virtuais conectados por chaves. As transferências de dados em um caminho virtual são totalmente independentes das transferências de dados em todos os outros caminhos virtuais. Para o programa, portanto, um caminho virtual dedicado não é diferente que um caminho físico dedicado.

Uma chave pode ligar um canal virtual de entrada a um ou mais canais virtuais de saída, isto é, difusão (ou "broadcasting") é possível. Existe uma fila dedicada para cada canal virtual de entrada, o que torna possível o escalonamento de transferência de dados através da chave localmente, independente de outras chaves. O uso de um canal físico entre chaves vizinhas é também escalonado local e dinamicamente. Detalhes da arquitetura da chave e demais características sobre a multiplexação de canais são discutidos em [10].

Duas observações devem ser feitas relativas à posição que a presente arquitetura ocupa no espaço de arranjos reconfiguráveis e redes de interconexão. A primeira é com respeito à complexidade de uma componente de chaveamento num arranjo reconfigurável e a segunda é a escolha da arquitetura de interconexão.

A alta reconfigurabilidade do arranjo é obtida às custas de uma chave complexa que é necessária para implementar o mecanismo de canais virtuais. Uma chave complexa pode não ser apropriada para arranjos de processadores simples, como é o caso da maioria dos arranjos reconfiguráveis propostos na literatura. Para arranjos de processadores poderosos como as células Warp, entretanto, uma chave complexa torna-se suportável e os benefícios podem superar os custos.

Redes de interconexão têm sido estudadas extensivamente na literatura [11]. Por exemplo, uma rede multi-estágios como a rede Omega pode prover interconexões flexíveis para um grande número de processadores, podendo também ser usada para construir um arranjo reconfigurável. Tal rede, entretanto, envolve linhas longas e a rede pode precisar ser reconstruída inteiramente quando o número de processadores é aumentado. Por outro lado, uma rede retangular é facilmente estensível. Por exemplo, a chave e a célula da Figura 1 podem ser construídas em conjunto e usadas como uma unidade para construir arranjos reconfiguráveis de vários tamanhos. Embora não com a mesma flexibilidade das redes multi-estágios, a rede retangular pode suportar melhor a transferência de alta largura de banda entre vizinhos.

3. RECONFIGURAÇAO

Na maioria dos trabalhos anteriores sobre arranjos bi-dimensionais reconfiguráveis [1, 2, 12], as estruturas lógicas implementadas no arranjo físico são também estruturas retangulares bi-dimensionais, e somente falhas de processadores são consideradas. O nosso esquema de reconfiguração pode suportar estruturas lógicas genéricas, não necessariamente arranjos retangulares. Além disso, em adição a falhas de processadores, também consideramos falhas de canais, chaves e "buffers" de E/S. O nosso esquema de reconfiguração distingue-se dos demais portanto nesses dois aspectos: a generalidade em suportar diferentes estruturas lógicas para programas e a consideração de falhas em diferentes tipos de componentes do arranjo físico.

3.1. Um Modelo Geral para Programas

Algoritmos paralelos para diferentes aplicações usualmente requerem diferentes estruturas lógicas de interconexão dos elementos de processamento. Algumas das estruturas lógicas usuais [13] incluem o arranjo linear ou uni-dimensional (filtros, transformada de Fourier discreta e convolução), arranjos bi-dimensionais (problemas em grafos usando matrizes de adjacência e programação dinâmica), e árvores (problemas de busca, avaliação de recorrências, etc.). Torna-se desejável portanto a possibilidade de mapear diferentes estruturas lógicas no mesmo hardware a fim de incrementar a programabilidade da máquina. Note-se que, mesmo sem considerar tolerância a falhas, tal arranjo reconfigurável por si só é desejável, por poder implementar um modelo geral de interconexão para programas.

A estrutura lógica de um programa é representada como um grafo orientado chamado grafo lógico, com os vértices ou nós representando processadores ou "buffers" de E/S e arestas representando as conexões entre eles. Dizemos que dois nós são vizinhos quando há uma aresta ligando-os.

3.2. Um Modelo Geral para o Arranjo Físico

Um dado arranjo físico pode ser de tamanho e forma arbitrários. Falhas podem ocorrer ao longo do tempo. O modelo de falhas supõe que, além de processadores e "buffers", chaves e canais físicos também podem falhar. Esta suposição é necessária pois a estrutura de interconexão do arranjo é constituída de componentes complexas.

A Figura 5 dá um exemplo de um arranjo físico com falhas onde apenas componentes não defeituosas são mostradas. Se um processador falhar, a chave à qual ele está ligado pode ainda rotear dados. Falhas de canais e chaves, entretanto, podem tornar inacessíveis processadores não defeituosos e assim inutilizando-os.

O algoritmo de reconfiguração deve levar em consideração tais problemas de conectividade.

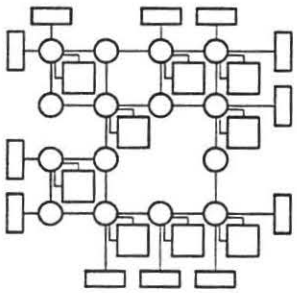

Figura 5. Arranjo físico representado como um grafo

No nosso modelo, o arranjo físico é representado como um grafo chamado grafo físico, com vértices ou nós representando processadores, chaves ou "buffers" não defeituosos, e arestas representando canais físicos não defeituosos.

3.3. O Algoritmo de Mapeamento

O problema de reconfiguração consiste no mapeamento do grafo lógico no grafo físico. Cada nó lógico do tipo processador ou "buffer" é alocado a um nó físico do mesmo tipo. Além disso, se há uma aresta de um nó lógico a outro, então deve haver um caminho entre os nós físicos correspondentes, passando através de uma série de nós de chaveamento. Cada tal caminho corresponde a um caminho virtual descrito na Seção 2.4.

Seja m o número de nós lógicos e seja n o número de nós físicos do tipo processador. Para simplicidade de explanação, a apresentação a seguir omite nós do tipo "buffer". Um algoritmo de mapeamento trivial seria tentar todas as alocações ("placement") possíveis de nós lógicos l_1, l_2, ..., l_m no grafo físico. Assim, l_1 pode ser alocado a qualquer dos n nós físicos p_1, p_2, ..., p_n (ver Figura 6). Para cada uma dessas alocações, l_2 pode por sua vez ser alocado a qualquer dos $n-1$ nós físicos restantes. Para cada alocação de l_2, se l_1 e l_2 são ligados por uma aresta, então devemos obter um caminho entre os nós físicos correspondentes. De maneira análoga, tratamos dos nós l_3, ... , l_m. Todas essas possíveis

alocações constituem uma árvore de alternativas, que denominaremos de árvore de alocação. A cada nível da árvore, um nó lógico está sendo alocado, com as ramificações denotando as possíveis alocações. O número total de nós terminais ou "folhas" na árvore de alocação é $n!/(n-m)!$. Cada caminho da raiz da árvore de alocação a uma folha corresponde a um possível mapeamento.

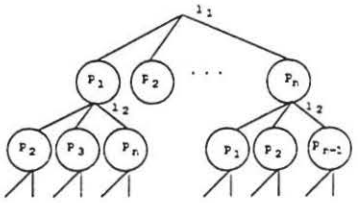

Figura 6. Uma árvore de alocação

Usamos várias métricas de availiação para medir a qualidade de um mapeamento. Uma tal métrica é o máximo número de canais virtuais usados num canal físico. Outras métricas incluem o comprimento total dos caminhos, o caminho mais comprido, etc. Um mapeamento pode ser escolhido baseado na sua qualidade com respeito a um conjunto de métricas de avaliação.

Uma busca exaustiva da árvore de alocação é de altíssimo custo computacional e não viável mesmo para tamanhos de grafos modestos. Ao invés disso, usamos um algoritmo de mapeamento baseado em busca heurística que possui as características seguintes.

Para cada nó lógico l_i, ordenamos os nós físicos começando com os candidatos mais promissores para a sua alocação. Esta ordenação se baseia na similaridade das propriedades de interconexão entre l_i e cada um dos nós físicos. Detalhes dessa heurística serão descritos na Seção 4. A ordenação é atualizada de maneira dinâmica, como veremos logo a seguir. Quando um nó lógico l_i é para ser alocado, tentamos primeiro o nó físico que é o candidato mais promissor, depois o segundo mais promissor, e assim por diante. Supondo que os ramos na árvore de alocação são expandidos da esquerda para a direita, então os ramos mais promissores são os da esquerda.

A alocação começa com um nó lógico l_s. Heurísticas são usadas para selecionar esse nó (como por exemplo escolhendo aquele nó com o maior número de descendentes para começar). Uma vez l_s está selecionado, a ordem para alocar os demais nós lógicos seguem a ordem de percurso por alargamento ("breadth-first") a partir de l_s.

Denotamos por l_i o nó lógico correntemente sendo alocado e por l_j um de seus nós vizinhos. Uma vez l_i está alocado no nó físico p_r, as ações seguintes são tomadas.

- Se l_j já foi alocado anteriormente, digamos, sobre p_s, então o caminho de roteamento mais curto entre p_r e p_s é determinado. (Vemos, portanto, que alocação e roteamento são realizados concorrentemente.) A métrica de avaliação é computada e usada para decidir se a busca deve continuar no ramo de busca corrente da árvore de alocação. Ramos de busca não promissores são assim podados de maneira eficiente.

- Se l_j ainda não foi alocado, então a sua lista de candidatos (nós físicos mais promissores) é re-ordenada como se segue: como l_j é um vizinho de l_i que foi alocado em p_r, os nós físicos próximos a p_r se tornam candidatos promissores.

Devido à imensidade do espaço das soluções, para grafos grandes, temos adotado vários critérios para parar a execução do algoritmo de mapeamento, antes que todas as soluções são examinadas. Esses critérios incluem a parada após alcançar qualquer nó folha na árvore de alocação, ou após a expansão de um dado número de nós da árvore de alocação.

4. As Heurísticas Usadas

Nesta seção justificamos por que escolhemos a abordagem heurística e mostramos as principais heurísticas usadas, relativas à obtenção dos nós físicos que são candidatos mais promissores para a alocação dos nós lógicos.

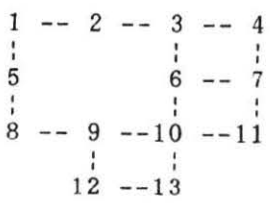

Figure 7a. Um grafo físico

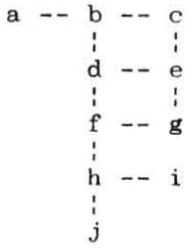

Figure 7b. Um grafo lógico

Vamos mostrar que o problema de mapeamento descrito neste artigo está na classe de problemas NP-completos. Claramente um mapeamento ótimo é obtido (com respeito, digamos, ao critério de minimizar o número máximo de canais virtuais em cada canal físico, ou minimizar o comprimento total dos caminhos usados no mapeamento) quando há uma correspondência um-para-um entre os nós e arestas do grafo lógico e os do grafo físico. Como um exemplo, consideremos os grafos físico e lógico das Figuras 7a e 7b. Um mapeamento perfeito é dado pelo seguinte.

Nó lógico	Nó físico
a	2
b	3
c	4
d	6
e	7
f	10
g	11
h	9
i	12
j	8

Assim sendo, um mapeamento ótimo é obtido se e somente se o grafo lógico é isomorfo a um subgrafo do grafo físico. O problema de mapeamento é portanto equivalente ao problema de isormorfismo de grafos que é NP-completo [14].

A nossa abordagem heurística a este problema consiste basicamente no seguinte.

- Para cada nó lógico, usamos heurísticas para ordenar os nós físicos para a sua alocação. Com isso, esperamos poder explorar primeiro os ramos mais promissores na árvore de alocação.

- Escolhemos a ordem por alargamento ("breadth-first") para alocar os nós lógicos l_1, l_2, ... e l_m, de tal maneira que em cada etapa de alocação (como foi descrito na seção anterior), somos capazes de realizar uma das seguintes ações:

  * Computar uma métrica de avaliação que é usada para decidir se devemos continuar a busca no ramo corrente da árvore de alocação ou podar o ramo corrente.

  * Reordenar os nós físicos mais promissores para alocação daqueles nós lógicos que serão alocados durantes as etapas de alocação seguintes.

Vamos agora descrever as heurísticas usadas para obter a ordenação dos nós físicos mais promissores para alocar um nó lógico.

Examinando os grafos das Figuras 7a e 7b, podemos ver, por exemplo, que um nó físico que é bom candidato para alocar o nó lógico e é o nó 7. Nossa decisão é baseada talvez na percepção que ambos os nós e e 7 possuem características de conectividade similares nos grafos. As heurísticas que iremos adotar tentam sugerir nós físicos promissores para alocar um nó lógico, examinando as características de conectividades dos nós nos grafos.

Para cada nó dado k num grafo (físico ou lógico), tentamos expressar as suas características de conectividade por meio de três sequências de inteiros, definidos como se segue.

Denotamos por v_i um nó à distância i do nó k. (Temos v_0 = k.) Seja d a distância do nó mais afastado do nó k. Definimos três sequências de inteiros:

e_i = nº de arestas entre v_{i-1} e v_i ,
n_i = nº de nós v_i ,
d_i = diferença e_i - n_i ,

onde i varia de 1 a d.

Assim para cada nó k, definimos a matriz de conectividade M_k

$$M_k = [\, m_{ij} \,] \text{ onde } m_{1j} = e_j,\ m_{2j} = n_j,\ m_{3j} = d_j, \text{ para } j = 1, \ldots, d$$

Podemos computar as matrizes de conectividade para todos os nós do grafo físico e grafo lógico. Usando o nosso exemplo das Figuras 7a e 7b, temos as matrizes P_u para nós físicos $\underline{u}$ e matrizes L_v para nós lógicos $\underline{v}$:

Grafo físico:

$$P_1 = \begin{matrix} 2 & 2 & 3 & 5 & 4 \\ 2 & 2 & 3 & 3 & 2 \\ 0 & 0 & 0 & 2 & 2 \end{matrix} \qquad P_2 = \begin{matrix} 2 & 3 & 4 & 5 & 2 \\ 2 & 3 & 3 & 3 & 1 \\ 0 & 0 & 1 & 2 & 1 \end{matrix}$$

$$P_3 = \begin{matrix} 3 & 4 & 5 & 4 \\ 3 & 3 & 4 & 2 \\ 0 & 1 & 1 & 2 \end{matrix} \qquad P_4 = \begin{matrix} 2 & 4 & 3 & 3 & 4 \\ 2 & 3 & 2 & 3 & 2 \\ 0 & 1 & 1 & 0 & 2 \end{matrix}$$

$$P_5 = \begin{matrix} 2 & 2 & 3 & 6 & 3 \\ 2 & 2 & 3 & 4 & 1 \\ 0 & 0 & 0 & 2 & 2 \end{matrix} \qquad P_6 = \begin{matrix} 3 & 7 & 4 & 2 \\ 3 & 5 & 3 & 1 \\ 0 & 2 & 1 & 1 \end{matrix}$$

$$P_7 = \begin{matrix} 3 & 4 & 3 & 4 & 2 \\ 3 & 2 & 3 & 3 & 1 \\ 0 & 2 & 0 & 1 & 1 \end{matrix} \qquad P_8 = \begin{matrix} 2 & 3 & 5 & 4 & 2 \\ 2 & 3 & 4 & 2 & 1 \\ 0 & 0 & 1 & 2 & 1 \end{matrix}$$

$$P_9 = \begin{matrix} 3 & 5 & 4 & 4 \\ 3 & 4 & 3 & 2 \\ 0 & 1 & 1 & 2 \end{matrix} \qquad P_{10} = \begin{matrix} 4 & 6 & 4 & 2 \\ 4 & 4 & 3 & 1 \\ 0 & 2 & 1 & 1 \end{matrix}$$

$$P_{11} = \begin{matrix} 2 & 5 & 5 & 2 & 2 \\ 2 & 4 & 3 & 2 & 1 \\ 0 & 1 & 2 & 0 & 1 \end{matrix} \qquad P_{12} = \begin{matrix} 2 & 3 & 3 & 4 & 4 \\ 2 & 2 & 3 & 3 & 2 \\ 0 & 1 & 0 & 1 & 2 \end{matrix}$$

$$P_{13} = \begin{matrix} 2 & 4 & 4 & 4 & 2 \\ 2 & 3 & 3 & 3 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{matrix}$$

Grafo lógico:

$$L_a = \begin{matrix} 1 & 2 & 3 & 3 & 2 \\ 1 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 0 \end{matrix} \qquad L_b = \begin{matrix} 3 & 3 & 3 & 2 \\ 3 & 2 & 2 & 2 \\ 0 & 1 & 1 & 0 \end{matrix}$$

$$L_c = \begin{matrix} 2 & 4 & 2 & 1 & 2 \\ 2 & 3 & 1 & 1 & 2 \\ 0 & 1 & 1 & 0 & 0 \end{matrix} \qquad L_d = \begin{matrix} 3 & 6 & 2 \\ 3 & 4 & 2 \\ 0 & 2 & 0 \end{matrix}$$

$$L_e = \begin{matrix} 3 & 4 & 2 & 2 \\ 3 & 2 & 2 & 2 \\ 0 & 2 & 0 & 0 \end{matrix} \qquad L_f = \begin{matrix} 3 & 5 & 3 \\ 3 & 4 & 2 \\ 0 & 1 & 1 \end{matrix}$$

$$L_g = \begin{matrix} 2 & 4 & 4 & 1 \\ 2 & 3 & 3 & 1 \\ 0 & 1 & 1 & 0 \end{matrix} \qquad L_h = \begin{matrix} 3 & 2 & 3 & 3 \\ 3 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{matrix}$$

$$L_i = \begin{matrix} 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 \end{matrix} = L_j$$

Para um dado nó $\underline{u}$ do grafo lógico, com matriz de conectividade $L_u = [\, l_{ij} \,]$, i = 1, 2, 3 e j = 1, ..., d, a "adequacidade" do nó físico $\underline{v}$ para alocar o nó lógico $\underline{u}$, com matriz de conectividade $P_v = [\, p_{ij} \,]$, é dada pelo número de vezes a condição seguinte é satisfeita:

$$l_{ij} \leq p_{ij} \text{ para } i = 1, 2, 3 \text{ e } j = 1, \ldots, d$$

Aplicando essa heurística nos nossos exemplos das Figuras 7a e 7b, obtemos a seguinte ordenação para os nós físicos pelas suas "adequacidades". Para simplicidade, mostramos apenas os primeiros nós físicos promissores.

Nó lógico	Nós físicos
a	2 8 13
b	3 9
c	4
d	6 10
e	7
f	6 9 10
g	3 6 9 10 11
h	3 6 9
i e j	1 12 8

Notem que os nós físicos do melhor mapeamento aparecem na maioria das vezes no começo. Isso significa que os ramos mais promissores são explorados em primeiro lugar na árvore de soluções, o que explica por que boas solucções têm sido obtidas mesmo quando paramos a execução do algoritmo de mapeamento sem examinar a árvore inteira.

5. COMENTARIOS FINAIS

O esquema de reconfiguração apresentado representa os arranjos lógicos e físicos como grafos quaisquer. Assim sendo, o método é bastante geral e o algoritmo pode de fato ser aplicado a arranjos físicos que não são arranjos retangulares. Essa generalidade no tipo dos arranjos físicos considerados, mais a suposição de que qualquer componente do sistema

pode falhar, distingue o presente trabalho dos anteriores na área de arranjos reconfiguráveis.

O algoritmo de reconfiguração tem complexidade exponencial no pior caso. Entretanto, devido às heurísticas usadas e à eficiente podação da árvore de soluções, obtivemos resultados bastante satisfatórios nas simulações realizadas. Milhares de casos foram simulados, em cada um dos quais geramos falhas arbitrárias nas diversas componentes de um arranjo físico e fizemos a reconfiguração após cada falha. Os grafos físicos usados foram de tamanhos variados, os maiores chegando a cerca de 400 nós. Os resultados dessas simulações, no que se refere à avaliação da arquitetura, ao desempenho do arranjo diante de falhas e à duração da sua vida útil, estão fora do escopo do presente artigo e encontram-se descritos em [15].

REFERENCIAS

[1] Kwang, J.H. e Raghavendra, C.S. VLSI Implementation of Fault-Tolerant Systolic Arrays. Proc. International Conference on Computer Design, October, 1986, pp. 110-113.

[2] Negrini, R., Sami, M. e Stefanelli, R. "Fault Tolerance Techniques for Array Structures used in Supercomputing". *Computer Magazine 19*, 2 (February 1986), pp. 78-87.

[3] Snyder, L. "Introduction to the Configurable, Highly Parallel Computer". *Computer 15*, 1 (January 1982), pp. 47-56.

[4] Annaratone, M., Arnould, E., Gross, T., Kung, H.T., Lam, M. Menzilcioglu, O. e Webb, J. "The Warp Computer: Architecture, Implementation and Performance". *IEEE Transactions on Computers C-36*, 12 (December 1987), pp. 1523-1538.

[5] Negrini, R., Sami, M.G., Stefanelli, R. Fault Tolerance Approaches for VLSI/WSI Arrays. Proc. IEEE Phoenix Conf. on Comp. and Communication, 1985, pp. 460-468.

[6] Singh, A.D. An Area Efficient Redundancy Scheme for Wafer Scale Processor Arrays. Proc. of International Conference on Computer Design, October, 1985, pp. 505-509.

[7] Bruegge, B., Chang, C., Cohn, R., Gross, T., Lam, M., Lieu, P. Noaman, A. e Yam, D. Programming Warp. COMPCON Spring '87, IEEE Computer Society, 1987.

[8] Kung, H.T. e Menzilcioglu, O. A General Switch Architecture for Fault-Tolerant VLSI Processor Arrays. Proceedings of International Symposium on VLSI Technology, Systems and Applications, May, 1987, pp. 211-217.

[9] Shombert, L.A. *Using Redundancy for Testable and Repairable Systolic Arrays*. Ph.D. thesis, Carnegie Mellon University, August, 1985.

[10] Kung, H.T. e Menzilcioglu, O. Virtual Channels for Fault-Tolerant Programmable Two-Dimensional Processor Arrays. Tech. Report CMU-CS-87-171, Carnegie Mellon University, December, 1986.

[11] Wu, C.L. e Feng, T.S. *Interconnection Networks for Parallel and Distributed Processing*. IEEE Computer Society Press, 1984.

[12] Lombardi, F., Negrini, R., Sami, M.G., Stefanelli, R. Reconfiguration of VLSI Arrays: a Covering Approach. Proceedings of 17th International Symposium on

Fault-Tolerant Computing, 1987, pp. 251-256.

[13] Kung, H.T. The Structure of Parallel Algorithms. Advances in Computers, Volume 19, New York, 1980, pp. 65-112.

[14] Garey, M.R. and Johnson, D.S. *Computers and Intractability - A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[15] Cohn, R., Kung, H.T., Menzilcioglu, O. e Song, S.W. A Highly Reconfigurable Array of Powerful Processors. Proceedings of SPIE Symposium, Vol. 975, Advanced Algorithms and Architectures for Signal Processing III, Society of Photo-Optical Instrumentation Engineers, August, 1988.