MÉTODOS EXPLÍCITOS POR AJUSTE DE EXPONENCIAL PARA SIMULAÇÃO ELÉCTRICA DE CIRCUITOS EM ARQUITECTURAS PARALELAS

LUÍS MIGUEL TEIXEIRA D' ÁVILA PINTO DA SILVEIRA

Licenciado em Engenharia Electrotécnica e de Computadores

pelo

INSTITUTO SUPERIOR TÉCNICO

UNIVERSIDADE TÉCNICA DE LISBOA

Tese submetida para satisfação parcial

dos requisitos do programa de mestrado

em

Engenharia Electrotécnica e Computadores

(Computadores)

INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Electrotécnica e de Computadores

LISBOA

Março 1989

Tese realizada sob a supervisão de

Doutor Eng^o Luís Manuel Ramos da Silva Vidigal

Professor Catedrático do Departamento de

Engenharia Electrotécnica e de Computadores

INSTITUTO SUPERIOR TÉCNICO

Resumo

Palavras chave - Simulação Eléctrica de Circuitos, Métodos Explícitos, Métodos Explícitos de Ajuste por Exponencial, Arquitecturas de Computadores de Processamento Paralelo, Projecto ESPRIT 1058, CINNAMON.

A Simulação de Circuitos é hoje em dia uma técnica indispensável na verificação e análise de um projecto de um Circuito Electrónico. O processo de simulação pode ser levado a cabo a vários níveis correspondentes ao detalhe utilizado na modelização do funcionamento dos dispositivos presentes num circuito. Através de um processo de simulação é possível detectar eventuais erros de concepção ou projecto de um circuito, o que atendendo à complexidade e custo dos circuitos actuais, representa uma minimização em termos humanos e financeiros do esforço colocado no desenvolvimento do circuito.

A Simulação Eléctrica de Circuitos modela os dispositivos a um nível de elevado detalhe. Baseando-se em equações diferenciais que descrevem as relações entre cargas, fluxos, correntes e tensões nos diversos elementos e na topologia de interligação dos mesmos, faz uso de ferramentas numéricas de elevada complexidade para resolver as equações daí resultantes. Atendendo à crescente complexidade e dimensão dos circuitos actualmente fabricados, bem como ao grau de perfeição das tecnologias utilizadas, a resolução deste tipo de problemas torna-se extremamente complexo e computacionalmente muito oneroso.

Novas metodologias têm sido propostas para uma resolução mais eficaz do problema. Estas em geral dividem-se entre o desenvolvimento ou refinamento de soluções existentes ou novas técnicas de resolução do problema numérico, ou ainda pela utilização de meios computacionais mais eficientes e poderosos. Este trabalho aborda uma alternativa mista, atendendo a que se refere a uma metodologia numérica recente, implementada em Arquitecturas de Computadores de Processamento Paralelo de informação. O código desenvolvido no âmbito deste trabalho assenta numa técnica de resolução numérica ainda em fase de consolidação e baseada em Métodos Explicítos de Aproximação por Exponencial, sendo dado particular destaque ao paralelismo inerente a este tipo de aproximações.

Em primeiro lugar é apresentada uma introdução à metodologia do problema, sendo discutidos os aspectos conceptuais, numéricos e arquitectónicos. Descreve-se em seguida o enquadramento em que o trabalho se inseriu e os objectivos que se propunha atingir. Posteriormente são descritos de forma mais detalhada as particularidades das implementações feitas, os algoritmos utilizados, bem como as opções tomadas na sua realização. Exemplos de utilização são apresentados, é feita uma análise do desempenho observado, bem como da aplicabilidade da metodologia proposta. Termina-se com um sumário crítico e uma discussão de possíveis desenvolvimentos na mesma área.

Abstract

Keywords - Electrical Circuit Simulation, Explicit Methods, Exponential Fitting Explicit Methods, Parallel Processing Computer Architectures, ESPRIT 1058 project, CINNAMON

Circuit Simulation is nowadays a well established technique for the verification of electronic circuits performance and functionality. The simulation process can be carried out at different levels corresponding to the detail used in modeling the functionality of the devices present in a circuit. Through the process of simulation, it becomes possible to detect errors in the design phase of a project, which, due to the complexity and cost of present circuits, represents a minimization of the financial and human effort dedicated to design development.

Electrical Circuit Simulation models devices at a very detailed level. Based on the constitutive equations of the elements and the topology of their interconnections, it uses numerical tools of high complexity to solve the resulting circuit equations. Due to the increasing complexity and size of presently fabricated circuits, and also to the degree of perfection attained by the technological processes used, the solution of such problems becomes extremely complex and computationaly very expensive.

New methodologies have been suggested in order to solve the problem more efficiently. These can in general be divided between the development or further refinement of existing numerical techniques, the development of new techniques, or the use of more powerful and efficient computational resources. The work described in this thesis is based on a mixed approach, since it refers to a new numerical methodology implemented on Parallel Processing Computer Architectures. The prototypes developed rely on a numerical technique not yet well understood, namely Explicit Exponential Fitting Approximations, with particular emphasis being given to the inherent parallelism of such approximations.

First an introduction to the methodology of the problem is presented, with a discussion of its conceptual, numerical and architectural aspects. Then, the environment in which the development took place is described together with the objectives of the project. Later, a more detailed description of the implementations is given, with respect to the algorithms used as well as the design options taken. Application examples are presented, and a thorough analysis of the observed performance as well as the applicability of such a methodology is conducted. Finally, a critical summary of the project is given with a discussion of possible developments in the same area.

Agradecimentos

Ao Professor Luís Vidigal pela sua amizade, orientação e encorajamento. A sua disponibilidade e o contínuo acompanhamento do trabalho por mim desenvolvido foram e continuam a ser essenciais para a minha formação em termos humanos, científicos e profissionais.

Ao Engenheiro Horácio Neto pela amizade, pela orientação e por todas as trocas de ideias em sessões que moldaram o meu conhecimento no campo e me permitiram, a cada passo, clarificar ideias quanto ao trabalho a ser desenvolvido.

Ao Engenheiro Paulo Bicudo pela extraordinária amizade que nos une e pela sua preciosa colaboração na realização de parte significativa deste trabalho.

Aos Engenheiros José Alexandre Grácio e Arlindo Oliveira pela salutar convivência e amizade construída ao longo de vários anos de trabalho em conjunto.

Ao Engenheiro João Paulo Silva e ao Paulo Flores pela ajuda prestada na realização de parte do trabalho que nesta tese consta.

A todo o grupo de Simulação do INESC e, em geral, ao grupo de VLSI, pelo ambiente de união, compreensão, trabalho e entreajuda, construído ao longo dos anos.

A meus pais, meu irmão e minha restante família, pela amizade testemunhada, e por uma vivência familiar que definiu tanto o meu carácter humano como a minha formação científica. Pelo diálogo e compreensão que me permitiu encontrar em casa o ambiente caloroso e repousante sempre necessário.

Finalmente à Júlia, pelo seu amor e paciência, pela amizade, dedicação e compreensão. Por todos os momentos em que a sua presença me incutiu a calma e a força necessária para enfrentar todos os obstáculos humanos e sociais que se me ofereceram.

Nota ao leitor

Apesar da necessidade compreensível desta tese dever ser escrita em português, na sua elaboração foram utilizados inúmeros termos da língua inglesa, particularmente os referentes a aspectos de ordem técnica cuja tradução para a língua Portuguesa não atingiu ainda um estado de total aceitação. Tal facto deve-se igualmente ao contexto em que o trabalho foi desenvolvido, pois tratando-se em parte de um trabalho de cooperação com outros parceiros europeus, tornou-se necessária a utilização de uma plataforma linguística comum que possibilitasse a troca de informação entre as partes envolvidas. Consciente da polémica que uma tal opção levanta, optou-se neste caso pela clareza do discurso, em sacrifício de critérios de pureza de estilo, embora apenas nos casos em que tal dificilmente seria evitável. Para facilitar a leitura, foi decidido incluir no início desta tese um glossário de termos cuja consulta deve ser encarada apenas como um auxiliar da compreensão do texto. O mesmo glossário contém um conjunto de abreviaturas e anacrónimos utilizados ao longo do texto, sempre que tal se justifica, por razões de clareza ou simplicidade. Foram igualmente incluídos no glossário, todos os termos, que apesar de traduzidos para Português, são de tal forma usuais que justificam ser mencionados no texto.

Terminologia

array computer : Computador formado a partir de um aglomerado de processadores semelhantes, em geral unidades funcionais, sob a dependência de uma unidade de controlo única. A estrutura de ligação entre os processadores pode ser muito variada.

behavioral, functional, logical, timing, switch, electrical, geometrical, device, process : níveis de representação e simulação de um circuito.

benchmark : programa de teste cuja execução permite medir determinadas características de desempenho de uma máquina.

bus : termo associado a um conjunto de linhas físicas pelas quais informação flui entre processadores, ou entre estes e a memória. Normalmente designado por barramento.

checkpoint strategy : Estratégia de Memorização que consiste na arquivação dos estados intermédios na evolução de um sistema dinâmico. Cada ponto corresponde ao estado do sistema num dado instante.

coarse-grain and fine-grain paralellism : correspondem à granularidade de um algoritmo, ou seja, à razão entre as necessidades computacionais e de comunicação que o mesmo apresenta.

DC analysis : análise do funcionamento em repouso.

execution overheads : computação que é necessário efectuar sem ligação directa com o algoritmo em causa.

data-flow computers : estruturas computacionais baseadas no controlo de programas através do fluxo de dados, em vez do fluxo de instruções vulgarmente encontrado em arquitecturas convencionais.

deadlock : situação indesejável num algoritmo em que por deficiência do mesmo, pode-se atingir uma situação não estável, da qual não é no entanto possível evoluir.

event-driven simulation : simulação controlada por acontecimentos. O controlo da simulação é feito de acordo com a marcação e resolução de acontecimentos.

layout : Especificação dos contornos de difusão, polisilício, metal, etc. que constituem o desenho do circuito integrado ou das máscaras utilizadas no seu fabrico.

load imbalance : desiquilíbrio nas distribuição da carga computacional entre os vários processadores num computador paralelo.

lock, unlock : Mecanismos de sincronização de tarefas em concorrência.

parser : processo ou programa que efectua uma análise de uma linguagem ao nível da sintaxe.

pipeline : Técnica que permite que uma sequência de instruções seja colocada numa fila de execução cuja entrada são os dados de um programa e de forma a que cada instrução é sucessivamente aplicada sobre o resultado da instrução anterior.

processor idling : existência de processadores em estados de espera durante a execução do algoritmo.

resource contention : contenção no acesso a recursos partilhados, numa máquina paralela.

rollback : Procedimento de Retrocesso ou processo que consiste em retroceder no tempo e recuperar o estado de um sistema num instante anterior. A consistência da informação geralmente implica a existência de uma *checkpoint strategy*.

scheduler : processo encarregado da gestão de um conjunto de recursos e que distribui os mesmos, entre os pedidos recebidos, consoante um algoritmo de prioridades indicado.

transient analysis : análise do funcionamento em resposta a transitórios nas entradas.

Abreviaturas

Todas as abreviaturas serão explicados na primeira referência que lhes fôr feita. Esta secção pretende somente constituir uma referência de consulta para o leitor, não sendo necessária a sua leitura prévia.

ALU : *(Arithmetic-Logic Unit)* Unidade Aritmética-Lógica, a unidade funcional de um processador.

ASTAP : Simulador de Circuitos ao nível Eléctrico (IBM).

CAD : *(Computer Aided-Design)* Projecto Assistido por Computador.

CAM : *(Computer Aided-Manufacturing)* Fabrico Assistido por Computador.

CE : *(Computational Element)* Elemento Computacional, constituído por uma unidade funcional e possivelmente por um banco de memória local.

CM : *(Connection Machine)* Computador Massivamente Paralelo.

CMOS : *(Complementary Metal-Oxide Semiconductor)* Tecnologia de fabricação de circuitos integrados. Tem associada uma metodologia característica.

CPU : *(Central Process Unit)* Unidade central de processamento.

FIFO : *(First-In First-Out)* Fila de espera cujo algoritmo indica que o primeiro elemento a sair é o que reside na fila à mais tempo (foi o primeiro a entrar).

INESC : Instituto de Engenharia de Sistemas e Computadores.

IP : *(Interactive Processor)* Processador interactivo encarregado do interface entre o utilizador e o sistema.

MIT : Massachusets Institute of Techonology, Boston, Massachusetts.

MOS : *(Metal-Oxide Semiconductor)* Tecnologia de fabricação de circuitos integrados.

PE : *(Processing Elements)* Unidade de Processamento, dependente de uma unidade de controlo externa, com capacidade de executar um conjunto de funções restrito.

RELAX : Simulador Eléctrico de Circuitos baseado em Relaxação de Ondas (Universidade de Berkeley).

SISD, SIMD, MISD, MIMD : *(Single or Multiple Instructions on Single or Multiple Data)* Taxonomia de Computadores Paralelos.

SPICE : Simulador de Circuitos baseado em Métodos Directos (Universidade de Berkeley).

TMC : Thinking Machines Corporation.

VLSI (LSI, MSI, SSI) : *(Very Large (Large, Medium, Small) Scale Integration)* Muito grande (grande, média, pequena) escala de integração.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo pretende ser uma introdução ao tipo de problemas abordado no contexto deste trabalho, bem como da metodologia que lhes está associada. Pretende-se com ele dar ao leitor uma panorâmica necessariamente breve dos problemas que levam à necessidade de utilizar processos de simulação, das diversas formas em que essa simulação pode ser levada a cabo e da necessidade actual de encontrar meios de tornar esse processo mais eficiente.

Começa-se por uma descrição geral do problema, em que se faz uma retrospectiva dos processos de verificação. Introduz-se depois o conceito de simulação em computador, explica-se a sua motivação e necessidade, abordam-se os vários níveis a que essa simulação pode ser efectuada e procura-se justificar dentre estes a importância da simulação ao nível eléctrico.

Apresentam-se depois alguns problemas relacionados com este tipo de simulação, baseados em dados correntes e em projecções que se pretendem justificar. Introduzem-se finalmente algumas soluções para o problema, com ênfase na utilização de meios computacionais mais eficientes, nomeadamente os baseados em Arquitecturas de Computadores de Processamento Paralelo.

1.1 Simulação de Circuitos

A simulação é um processo destinado a fornecer informação sobre o funcionamento de um sistema sem actuar directamente sobre uma realização física do mesmo. A simulação é utilizada a vários níveis e numa grande diversidade de situações, sendo muitas vezes a única forma de obter informação sobre um dado comportamento ou função.

Trata-se de um processo utilizado para estudar a interacção entre os componentes fundamentais de um sistema, numa fase anterior ao seu fabrico ou construção, servindo simultaneamente de ferramenta de síntese e análise. Torna-se assim possível, através de simulação, corrigir ou alterar determinados factores de forma a tornar um projecto correcto e viável.

Associado ao processo de simulação existe todo um conjunto de técnicas destinadas a modelar o comportamento do sistema a ser simulado. A correcção e precisão dos modelos a utilizar condiciona em grande parte os resultados que se podem obter. De facto, se se modelar de forma incorrecta ou inconsistente o comportamento de um sistema, torna-se extremamente difícil extrapolar dos resultados obtidos qual será o comportamento do sistema real, da mesma forma que se torna praticamente impossível extrair a informação necessária de forma a alterar os parâmetros do sistema para que o mesmo se passe a comportar de forma a satisfazer as especificações. Esta questão não sendo um problema da simulação em si, está-lhe inerentemente associada. Trata-se aqui de construir um modelo que possibilite que o processo de simulação seja conduzido, de forma a reproduzir tão fielmente quanto possível as características e a dinâmica de um sistema real, de forma a que o grau de confiança nos resultados obtidos seja elevado.

A outra componente fundamental do processo de simulação, além da correcta modelação dos parâmetros do sistema, prende-se com a interacção entre os mesmos e as leis que regem essa interação. Este processo pode ser encarado de novo como um processo de modelação, agora não ao nível individual do comportamento de cada entidade de um sistema, mas ao nível da interacção das suas várias componentes.

1.1.1 Objectivos da Simulação

A simulação em computador do comportamento de um sistema integrado tem como objectivo não só a verificação da correcção do mesmo, mas igualmente possibilitar uma medida do desempenho do sistema real. Através da simulação torna-se possível detectar eventuais erros de concepção de um dado sistema bem como erros introduzidos na sua implementação. Da mesma forma torna-se possível, avaliando o desempenho do sistema simulado, determinar em que medida ele satisfaz as especificações pretendidas e possivelmente de que forma a implementação pode ser alterada para o desempenho ser melhorado.

Os objectivos do processo de simulação são assim duplos. Por um lado uma característica de análise, de verificação e de teste de um sistema. Por outro lado uma característica de síntese associada a uma avaliação de desempenho.

Embora parcialmente destinado a minimizar os custos de desenvolvimento de um dado projecto, o próprio processo de simulação tem, no entanto, vários custos associados: não só é necessário estabelecer um modelo do sistema a simular e verificar a correcção desse modelo, alterá-lo ou melhorá-lo, como há todo um custo computacional de simulação que pode ser extremamente elevado. Se a primeira destas componentes pode ser encarada como uma despesa fixa e até certo ponto desprezável, dado que o custo de desenvolvimento de um simulador é marginal face à utilização que o mesmo poderá ter, já o custo computacional de simulação é uma função crescente da complexidade de um projecto. O fabrico de tais sistemas é, no entanto, tecnologicamente muito difícil e oneroso, e a complexidade dos projectos desenvolvidos cresce a um ritmo alucinante, o que aumenta os custos de fabrico e diminui a probabilidade de se obter um produto funcionalmente correcto, sem uma verificação prévia. Assim, os benefícios obtidos através de simulação excedem largamente qualquer custo computacional que se possa imputar ao processo.

A simulação de circuitos integrados tem assim uma utilidade fundamental uma vez que numa perspectiva global permite minimizar os custos de desenvolvimento de um projecto fornecendo ainda informação sobre o desempenho futuro do mesmo. É igualmente útil na medida em que se torna extremamente simples fazer projecções sobre o comportamento de um circuito se um determinado conjunto de condições se verificar, tornando possível monitorizar todos os componentes de um sistema e observar todos os seus estados internos.

Um outro factor torna a simulação de circuitos integrados numa ferramenta imprescindível de projecto. É possível na simulação introduzir determinadas características intrínsecas ao processo de fabrico e à tecnologia usada, que são impossíveis de reproduzir num protótipo feito à custa de componentes discretos. Este factor tem-se tornado ainda mais importante com o aperfeiçoamento da tecnologia, de tal forma que o grau de confiança nos resultados obtidos através de um protótipo discreto é, em geral, extremamente baixo.

Salienta-se, no entanto, que o processo de simulação não é isento de erros. De facto, se por um lado o detalhe utilizado na modelação de um sistema condiciona a precisão e a confiança nos resultados a obter da simulação do mesmo, também é verdade que é geralmente impossível modelar todos os parâmetros de um sistema e a complexidade das interacções entre eles. O processo de simulação tem assim associado a si um grau de confiança que, pelo menos, deve ser claramente entendido de forma a poder extrair-se dos seus resultados o seu verdadeiro significado.

1.1.2 Níveis de Simulação

Associado a qualquer processo de simulação está um modelo do sistema a simular. O modelo utilizado deve traduzir de certa forma as características dos dispositivos utilizados e condiciona a forma como se equacionam as interações entre esses dispositivos. Qualquer modelo deve fornecer de uma forma mais ou menos precisa o comportamento das variáveis de interesse no sistema em estudo.

A precisão dos resultados obtidos através da simulação é condicionada pelos modelos utilizados na própria simulação. Este ponto é de extrema importância, pois estabelece, ainda que de forma não rigorosa, uma relação directa entre a validade e profundidade dos resultados de uma simulação e os modelos dos dispositivos utilizados na mesma.

A escolha de um dado modelo para representar o comportamento de um dispositivo estabelece assim limites bem definidos quanto à interpretação dos resultados a obter. Antes de se iniciar a simulação de um projecto, deve-se definir de forma precisa a forma e detalhe dos resultados que se pretende obter e escolher um modelo para os dispositivos que permita validar as estimativas obtidas e extrapolar dos resultados as conclusões pretendidas, com perfeito conhecimento do erro que esses resultados ou projecções encerram.

Na simulação de circuitos em computador é vulgar utilizar-se um conjunto quantificável de modelos para os dispositivos, que em geral se convencionou denominar níveis de representação e simulação. Um modelo tem assim associado uma representação e um conjunto de blocos básicos. Os circuitos integrados podem ser descritos e simulados a diferentes níveis de abstracção dentre os quais os mais usuais são:

- nível de comportamento
- nível funcional
- nível lógico
- nível temporal
- nível de interruptor
- nível eléctrico
- nível geométrico
- nível de dispositivo
- nível de processo.

O modelo ao nível de comportamento (designado na literatura por *"behavioral model"*) corresponde ao nível de maior abstracção, no qual um sistema é fundamentalmente descrito através de um algoritmo que lhe está subjacente e do fluxo de dados que proporciona, não sendo sequer necessário um conhecimento da estrutura da implementação do projecto. Este nível de representação corresponde a um nível arquitectural ou de especificação e pode servir de base para um projecto cujas unidades componentes serão posteriormente simuladas a um nível de maior detalhe. Tal tipo de metodologia é usualmente conhecida como aproximação de alto a baixo (ou aproximação *"top-down"*).

O modelo ao nível funcional descreve as interacções entre blocos ou unidades de complexidade variável mas com uma função bem definida. Descreve o fluxo dos dados entre as diversas unidades funcionais e o processo como estas tratam esses dados. Como exemplo de unidades a este nível temos básculas, registos, amplificadores, conversores, etc. O modelo ao nível funcional é geralmente o segundo passo na hierarquia de uma aproximação de alto a baixo e corresponde a uma formalização mais concreta das especificações e conceitos descritos e testados ao nível de comportamento. A este nível é já possível ter uma ideia mais concreta da viabilidade de um algoritmo, dado que este pode ser esquematizado em termos de blocos funcionais existentes.

O modelo ao nível lógico (vulgarmente designado por *"gate-level model"*) é utilizado apenas em circuitos com um comportamento digital (em que as variáveis de interesse assumem um conjunto de valores discreto) e tem como bloco básico a porta lógica. De novo considerando uma aproximação de alto a baixo, corresponde a uma primeira concretização de cada uma das unidades descritas ao nível funcional. A este nível o projectista pode não só continuar o processo de validação de um projecto, como obter uma primeira e muito vaga informação do desempenho futuro do mesmo. O modelo de nível lógico é tradicionalmente o mais utilizado na simulação de projectos de grande dimensão, por razões de simplicidade e históricas.

O nível eléctrico (*"electric-level"* ou *"circuit-level"*) é um modelo já de elevada complexidade, em que um circuito é descrito em termos de elementos básicos do tipo resistências, condensadores, transistores, etc. A cada um destes dispositivos é associada uma formulação que descreve as suas equações constitutivas, que no caso dos elementos mais complexos pode ser extremamente elaborada e de elevado custo computacional. A simulação a este nível permite obter informação temporal mais detalhada, bem como medir já com uma certa precisão outro tipo de características de um circuito, como a dissipação de potência, imunidade ao ruído, níveis de tensão, etc. Devido à sua complexidade, a simulação a este nível é usualmente feita apenas para blocos de pequena ou média dimensão.

O nível temporal (*"timing-level"*) tem merecido muita atenção em anos recentes [Lin86] [Meersch86] [Tsao85]. Trata-se de um nível de representação próximo do eléctrico, utilizando o mesmo tipo de componentes, mas com modelos bastante mais simples. A formulação das equações de um circuito e sua resolução é feito de forma muito diversa, empregando técnicas próprias do nível temporal, e pretende-se ser capaz de obter dados sobre o desempenho de um circuito em termos temporais. Trata-se de um nível de simulação em que se torna possível, com consideravelmente menos esforço, obter dados anteriormente só possíveis de obter simulando ao nível eléctrico, com a desvantagem desses dados serem bastante menos fiáveis. É em geral utilizado para circuitos de maiores dimensões para os quais se pretende uma informação de carácter global, pagando-se para tal o preço de uma menor precisão.

O modelo ao nível de interruptor (*"switch-level"*) utiliza técnicas extremamente recentes [Bryant80] e situa-se num nível de complexidade intermédio entre os níveis lógico e eléctrico. Introduzido como extensão do nível lógico, tem associado uma precisão superior à daquele, mas ainda assim inferior ao necessário para determinado tipo de aplicações, nomeadamente analógicas, ou mistas analógicas-digitais.

O modelo ao nível geométrico está relacionado com a forma física do circuito sendo uma descrição das máscaras utilizadas no fabrico desse circuito, nomeadamente no seu *layout*.

Finalmente o modelo ao nível de dispositivo é o modelo de maior complexidade em que se pretende modelar cada elemento em termos das leis físicas que o caracterizam, como sejam a concentração de portadores, os seus perfis, a difusão dos mesmos, etc. Devido à sua elevada complexidade, este nível de representação é apenas utilizado para modelar um número extremamente reduzido de dispositivos em cada simulação. Igualmente, dado que envolve um grande conhecimento dos processos de fabrico, não é um modelo vulgarmente utilizado por um projectista.

Um grande esforço e atenção têm sido devotados a um outro tipo de representações que estão associados à simulação multi-nível [Odryna86ii]. No processo de simulação multi-nível de um sistema, este é subdividido em vários blocos, que podem ser simulados a níveis distintos. Este tema é de extrema importância, pois a interligação automática de simuladores a níveis diferentes pode simplificar grandemente o projecto de simulação de um sistema, por escolha correcta do nível adequado à análise de cada um dos blocos.

O trabalho desenvolvido nesta tese foca quase exclusivamente o nível eléctrico de representação e simulação. Este nível é de uma importância capital, dado que é o único processo preciso, fiável e tecnologicamente independente que permite descrever o comportamento de um circuito integrado. Com a crescente complexidade tornada possível por uma tecnologia em rápido crescimento, torna-se ainda mais necessário verificar as partes cruciais de um projecto com elevada precisão, dado os custos inerentes ao fabrico de tais circuitos. Apenas ao nível eléctrico é possível simular um circuito de forma a não só avaliar a sua correcção, mas igualmente medir com erro aceitável o seu desempenho [Yeh88].

1.2 Aceleração da Simulação de Circuitos

Com o desenvolvimento das tecnologias de fabrico, em particular dos circuitos MOS, a dimensão dos sistemas hoje em dia encontrados num circuito integrado é de tal forma elevada que os custos, crescentes, associados à sua simulação tendem a não ser compensados pelos benefícios que daí se podem extrair. Efectivamente o custo computacional da simulação tende a crescer de forma sobre-linear com a dimensão de um circuito, o que torna, a médio prazo, incomportável a simulação de sistemas completos, mesmo ao nível lógico.

Os simuladores de circuitos tradicionais, desenvolvidos nos princípios dos anos 70, tinham como objectivo fundamental a simulação de circuitos com poucas dezenas de transistores. Com a necessidade de simular circuitos com uma dimensão e complexidade crescentes, esses programas foram sendo refinados e remodelados de forma a a sua execução ser acelerada. Tornou-se no entanto claro que este processo é claramente limitado, pelo que há que procurar novas respostas à questão da necessidade de acelerar o processo de simulação ao nível eléctrico.

Nos últimos anos, essas respostas têm sido tentadas quer sob a forma de novos algoritmos quer sob a forma de meios computacionais mais eficientes. O processo de desenvolvimento de novos algoritmos tem sido constantemente abordado tendo-se conseguido alguns avanços no campo, embora de forma não decisiva. Recentemente, no entanto, um conjunto de novas técnicas, nas quais se enquadram algumas das descritas nesta tese, tem vindo a ser desenvolvido e utilizado com resultados muito satisfatórios. Este tema não está de forma alguma encerrado, sendo alvo de acesa pesquisa que se traduz em programas de investigação com financiamentos muito elevados.

A procura incessante de meios computacionais mais eficientes e poderosos, tem sido uma constante associada a todo este processo. A complexidade actual de um sistema integrado aproxima-se da cifra dos 10^6 dispositivos [Dally86], e as necessidades computacionais da simulação de tal tipo de circuitos começam a exceder a capacidade dos computadores actuais. Dois tipos de alternativas são em geral seguidos: a construção de **Máquinas de Simulação**, computadores especificamente destinados à resolução de um problema [Blank84] [Dally] [Hahn85], ou a utilização de **Arquitecturas de Processamento Paralelo**, computadores com mais de uma Unidade Central de Processamento (CPU).

Máquinas de simulação ou computadores específicos são geralmente uma solução eficiente embora com algumas desvantagens. Em particular a sua especificidade torna-os practicamente inúteis como ferramenta computacional para problemas que não aquele para o qual foram desenhados, e correm o risco de se tornar obsoletos num prazo relativamente curto, dependendo da evolução da tecnologia e das técnicas de simulação.

O processamento paralelo surge então como uma alternativa extremamente interessante para a resolução deste tipo de problemas. A utilização de Computadores baseados em Arquitecturas de Processamento Paralelo de uso geral, tem a vantagem da sua não especificidade os tornar utilizáveis para a resolução de problemas muito diversos, de serem mais facilmente produzidos, de preço mais reduzido e com menor probabilidade de se tornarem completamente inúteis caso novas metodologias de simulação sejam introduzidas, ou a tecnologia se altere.

A utilização de Computadores Paralelos coloca no entanto o problema do desenvolvimento de algoritmos que possam tirar partido do paralelismo oferecido. A quase totalidade dos algoritmos desenvolvidos até ao início da década de 80 baseava-se em máquinas uniprocessadoras, com uma arquitectura de Von Neumann, e muitas vezes não pareciam capazes de explorar as potencialidades de uma máquina paralela. Torna-se assim necessário desenvolver novos algoritmos, ou alterar os existentes, de forma a que o paralelismo seja eficientemente aproveitado. Este processo criou, de facto, toda uma nova área de investigação, dado que rapidamente se começou a verificar que a eficiência de uma implementação paralela de um dado algoritmo é criticamente dependente não só da configuração particular da máquina utilizada, mas em grande parte da forma como o paralelismo é explorado ao nível do algoritmo.

É fundamentalmente nesta área de pesquisa que o trabalho a que esta tese se reporta foi efectuado. De notar que se trata de uma área bastante recente e portanto num processo ainda de inovação mais do que de amadurecimento, especialmente porque o próprio desenvolvimento de Computadores Paralelos está ainda nos seus primeiros passos.

1.3 Organização da Tese

Esta tese está dividida em 6 capítulos em que cada um encerra um tema específico numa sequência que foi julgada a mais lógica. Após a introdução feita neste primeiro capítulo, em que foram introduzidos os conceitos básicos gerais da simulação de circuitos e motivado o trabalho efectuado de pesquisa de algoritmos para implementação de simuladores de circuitos em Arquitecturas de Processamento Paralelo, os capítulos seguintes focam temas mais concretos na sequência do referido anteriormente.

O próximo capítulo aborda o tema de Arquitecturas de Processamento Paralelo, descrevendo temas tão gerais como as opções arquitectónicas existentes ou as máquinas comercias já desenvolvidas. Este capítulo pode ser visto como uma introdução a uma das duas partes fundamentais desta tese e aborda alguns assuntos de importância fulcral para o correcto entendimento do trabalho aqui descrito.

O terceiro capítulo é igualmente de carácter introdutório, mas aborda o contexto em que este trabalho particular está inserido. Nele se descreve o esforço desenvolvido em particular pelo Projecto de Simulação de Circuitos do INESC, na pesquisa de novas técnicas e algoritmos para a resolução do problema. Igualmente se apresentará o material necessário a uma melhor compreensão das técnicas e processos envolvidos em Simulação Eléctrica de Circuitos em Computador.

O quarto capítulo detalha o trabalho realizado no contexto desta tese. Nele se descrevem os passos tomados no desenvolvimento dos protótipos que se apresentam, os algoritmos em que se baseiam, as diferenças entre eles, as ideias em que assentam e a forma como foram desenvolvidos.

No quinto capítulo apresenta-se um conjunto de resultados que se tenta aglomerar num todo que permita uma visão de conjunto sobre as técnicas estudadas. Nele se procura igualmente agrupar os resultados obtidos de uma forma coerente e que permita extrair conclusões com algum valor.

No sexto capítulo sumarizam-se os resultados obtidos e critica-se, de forma construtiva, o trabalho realizado, procurando determinar o rumo a dar a novos esforços a efectuar.

Finalmente os apêndices, contêm um conjunto de informação de utilidade para quem se pretender debruçar de forma precisa sobre o trabalho desenvolvido, não sendo no entanto indispensáveis à correcta compreensão dos conceitos envolvidos. Uma bibliografia de referências é apresentada por último, contendo menção a extenso número de livros e artigos.

Capítulo 2

Processamento Paralelo

O **Processamento Paralelo** emergiu nos últimos anos como uma solução possível e interessante para a incessante procura de meios computacionais mais eficientes e de melhor desempenho [Asbury85] [Hancock86].

À medida que nas sociedades tecnologicamente mais avançadas o uso de computadores se torna rotineiro e se extende a um sem número de campos nas áreas de administração, medicina, indústria, comércio, defesa, etc, um interesse crescente é dedicado ao desenvolvimento de máquinas mais poderosas e mais baratas.

Os pontos fulcrais na procura de maior poder computacional são maior velocidade de processamento, melhor desempenho, maior fiabilidade e custos menores. Estes quatro pontos são em certa medida mutuamente exclusivos e um compromisso tem de ser assumido para tornar possível atingir os objectivos propostos [Delgado].

As arquitecturas de computadores mais avançadas, estão actualmente centradas em torno do conceito de Processamento Paralelo, dado que esta parece ser a solução mais promissora e viável para os quatro objectivos mencionados anteriormente. Os computadores paralelos representam no entanto, conceptualmente, um afastamento tão radical das arquitecturas convencionais, que não é fácil definir uma taxonomia pela qual possam ser unívoca e claramente classificados.

Os computadores paralelos não são no entanto um conceito recente. De facto, a ideia de resolver simultaneamente vários problemas para diminuir o tempo de desenvolvimento de um dado projecto, tem vários anos. No entanto a maioria das tentativas sérias de abordar o problemas depararam com dificuldades de vária ordem, técnica, conceptual ou financeira. Acresce que ao problema arquitectónico em si, se junta o de eficazmente aproveitar as possibilidades de um tal tipo de arquitecturas. Tal implica o desenvolvimento de um ambiente de programação passível de permitir a um utilizador comum o desenvolvimento de ferramentas capazes de utilizar todas as capacidades postas à sua disposição.

O desenvolvimento de ambientes de programação em arquitecturas paralelas, ou de ferramentas utilizáveis num tal tipo de máquinas, implica no entanto um conhecimento aprofundado dos tipos de paralelismo possíveis e das opções arquitectónicas disponíveis.

2.1 Arquitecturas de Processamento Paralelo

O desenvolvimento de um modelo computacional para **Processamento Paralelo** é uma necessidade não apenas do ponto de vista teórico, mas igualmente para providenciar uma base de desenvolvimento de algoritmos. A criação de um tal modelo tem no entanto esbarrado com consideráveis dificuldades devido à grande liberdade e diversidade na definição das possíveis arquitecturas.

A classificação mais conhecida para computadores paralelos [Flynn66] (apresentada na tabela 2.1), baseia-se no fluxo de instruções que um computador executa simultaneamente e no fluxo de dados sobre os quais actua. Tal taxonomia leva em geral a uma divisão em 4 categorias: as máquinas capazes de executar apenas uma instrução sobre um único conjunto de dados, **SISD** *(Single-Instruction on Single-Data)*, aquelas que executam apenas uma instrução sobre vários dados ao mesmo tempo, **SIMD** *(Single-Instruction on Multiple-Data)*, as que executam mais do que uma instrução em simultâneo sobre um único grupo de dados, **MISD** *(Multiple-Instruction over Single-Data)* ou sobre vários dados ao mesmo tempo, **MIMD** *(Multiple-Instruction over Multiple-Data)*.

	Dados Únicos	**Múltiplos Dados**
Instrução Única	SISD	SIMD
Múltiplas Instruções	MISD	MIMD

Tabela 2.1: Taxonomia de Flynn para Arquitecturas de Processamento Paralelo

Tal taxonomia, embora a mais geral, tem sido frequentemente posta em causa devido à grande diversidade de arquitecturas propostas que não se enquadram verdadeiramente apenas numa categoria. No entanto, é do consenso geral que as máquinas SISD são os computadores convencionais, com uma estrutura de Von-Neumann tradicional baseada num único processador que executa uma sequência de acções sobre dados existentes num banco de memória. A categoria MISD é uma categoria que pode ser considerada vazia, dado que dependendo do tipo de argumentos utilizados, estas máquinas podem ser consideradas como SISD.

Como foi dito, além da taxonomia apresentada, existem uma variedade de outras, dependendo do autor e dos parâmetros pelos quais se diferenciam as arquitecturas paralelas [Hockney88]. Alguns autores consideram o factor preço como determinante e apresentam uma divisão baseada em tabelas de dupla entrada com escalas de preços e arquitectura básica [Manuel86]. Outros autores preferem uma divisão mais estrutural baseada nos conceitos de paralelismo explorados em cada arquitectura [Hwang84].

Computadores Paralelos são máquinas que se baseiam em processamento simultâneo ou paralelo, seja paralelismo temporal, paralelismo espacial ou paralelismo assíncrono. Um sistema que se baseie em paralelismo temporal tem de sobrepôr a execução de várias instruções, de certa forma providenciando um *pipeline* em que as instruções fluem. A obtenção de paralelismo espacial requer a existência de múltiplas unidades de processamento operando sincronamente, executando a mesma instrução simultaneamente. Paralelismo assíncrono está relacionado com a existência de múltiplos processadores de capacidades semelhantes partilhando o acesso seja a memória, a entradas-saídas ou a outros periféricos [Paker83].

As três formas de paralelismo mencionadas não são de forma alguma exclusivas. De facto, muitas das máquinas existentes exploram mais do que um tipo de paralelismo. Por exemplo, muitas das máquinas paralelas já construídas têm alguma forma de *pipeline* de instruções podendo ser, no entanto, construídas à base de uma matriz de processadores. Além das formas de paralelismo exploradas em cada arquitectura, a estrutura da rede de interligação entre os vários processadores é também um factor distintivo de uma arquitectura.

2.1.1 Paralelismo Temporal

A formação de um *pipeline* está directamente ligada aos passos requeridos no processo de execução de uma instrução num computador digital, o que geralmente envolve quatro tarefas distintas: ler a instrução, descodificá-la, ler os operandos e executar a instrução descodificada sobre os operandos desejados. Se cada uma destas operações é considerada indivisível[1] e assignada a um estágio da *pipeline*, não há verdadeiramente necessidade de uma instrução completar os quatro estágios antes de a próxima instrução se iniciar. É

[1] A terminologia atómica para designar uma operação indivisível é igualmente corrente.

assim possível sobrepôr instruções bastando para tal colocar os vários estágios do processo em cascata. Como exemplo, a figura 2.1 apresenta um conjunto de operações elementares cuja execução se pode efectuar em sequência, como na figura 2.2, ou aproveitando paralelismo temporal, através de um *pipeline*, como indicado na figura 2.3[2].

```
x = a + b
y = c * d
z = sqrt(e)
u = exp(f)
```

Figura 2.1: Conjunto de operações elementares

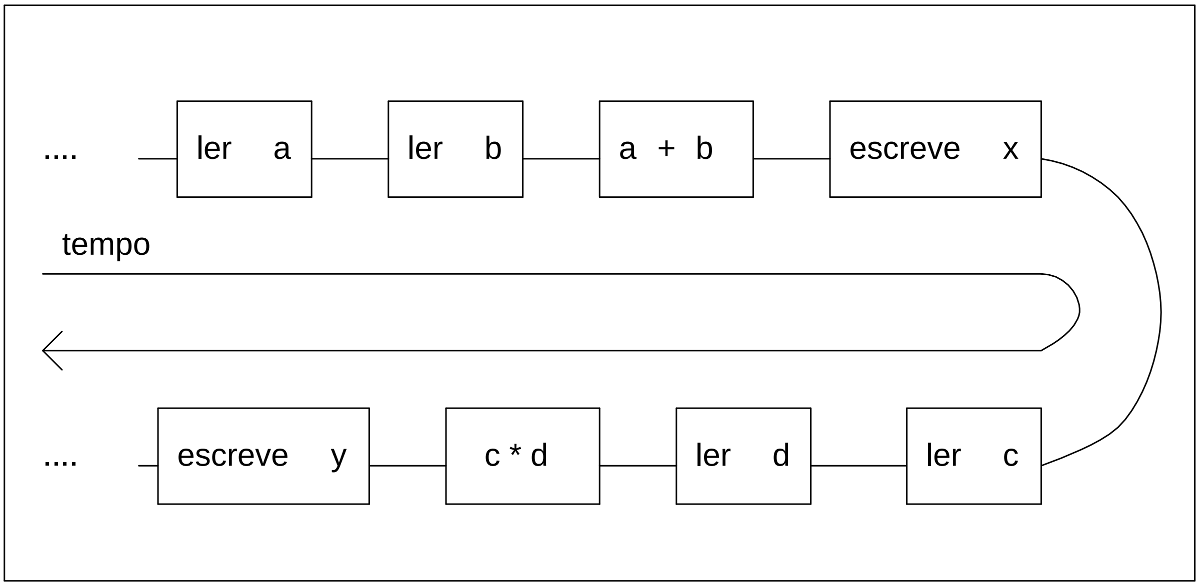

Figura 2.2: Execução sequencial das operações indicadas na figura 2.1

Em qualquer instante, cada estágio do *pipeline* lida com uma instrução, e um ou mais dos estágios necessários para a execução dessa instrução está a processar-se. Este conceito pode ser estendido a um nível superior da hierarquia de programação podendo cada estágio de um *pipeline* encarregar-se de um conjunto de operações e não apenas de uma operação

[2]O *pipeline* apresentado neste exemplo tem quatro estágios. No caso geral o número de estágios de um *pipeline*, ou é determinado pela arquitectura da máquina, ou pode ser fixado pelo utilizador.

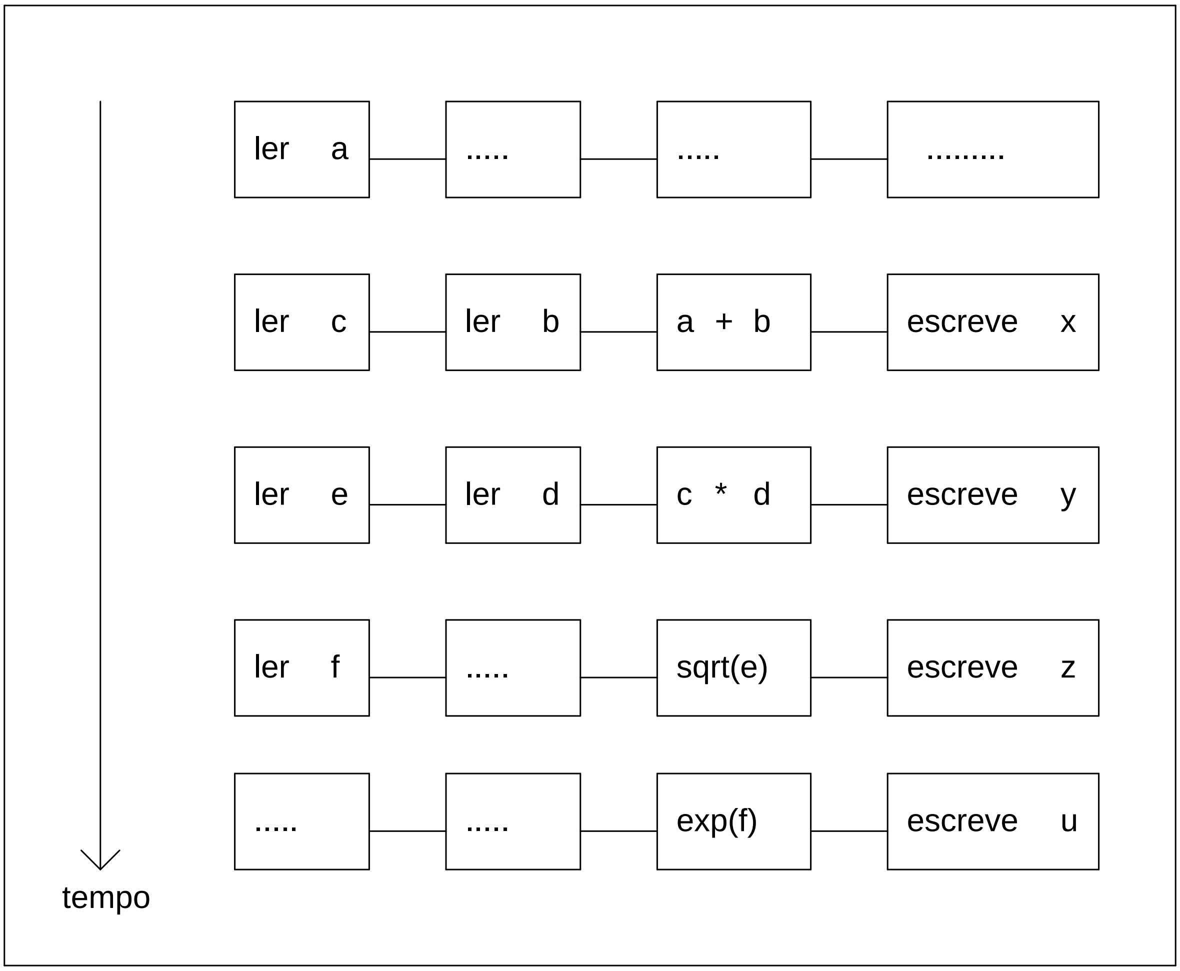

Figura 2.3: Execução em paralelo das operações indicadas na figura 2.1 utilizando um *pipeline* de quatro estágios

atómica. É claro, portanto, que o comprimento do *pipeline* pode ser qualquer, pelo que se torna evidente que num *pipeline* de n estágios é possível, teoricamente obter um aumento de velocidade de n. Note-se que os vários estágios do *pipeline* não necessitam de ter a mesma duração, pelo que o mais lento dita a duração do ciclo do *pipeline*, de forma a evitar colisões. A operação do *pipeline* é portanto sincronizada por um relógio comum.

As questões fundamentais no desenho de um *pipeline* de um computador são fundamentalmente o sequenciamento das tarefas, a prevenção de colisões no acesso a dados comuns entre os vários estágios, o controlo da congestão, a reconfiguração e a resolução de conflitos. Note-se que conflitos no acesso a memória, dependência entre os dados, interrupções ou saltos dentro da execução do mesmo são sempre factores de abrandamento da velocidade de execução ou causa de congestão no *pipeline* e devem ser considerados com grande cuidado. Em particular, todas as operações que envolvam o reconfiguramento do *pipeline*[3] ou a execução de saltos dentro de um programa, implicam a necessidade de esvaziamento do *pipeline*, dado que a sequência de endereçamento de dados pode não ser a sequência lógica de execução.

A utilização de paralelismo temporal pelo estabelecimento de um *pipeline* é uma técnica muito usada para acelerar a velocidade de execução de uma unidade de processamento, ou para a realização de operações repetitivas. Como foi já dito, a maioria das máquinas correntemente construídas utilizando *pipeline*, ou são uniprocessadores em que a unidade central utiliza *pipelining* ao nível das operações atómicas, ou exploram igualmente outra forma de paralelismo. *Pipelining* é portanto uma técnica vulgar em supercomputadores[4] sendo muitas vezes considerado apenas como um mecanismo de aceleração de execução em vez de uma característica arquitectónica básica.

2.1.2 Paralelismo Espacial

Matrizes de Processadores, vulgarmente designados por *Array Computers* exploram paralelismo espacial através de múltiplas unidades aritmético-lógicas *(ALU)* (usualmente designadas por **Elementos Processadores**, ou **PE**'s, *Processing Elements*) trabalhando sincronamente em paralelo. Uma unidade de controlo externa encarrega-se do direccionamento correcto dos dados entre os vários PE e do controlo da rede de interligação entre aqueles. Instruções vectoriais são difundidas para a memória local de cada PE, sendo os dados lidos e armazenados localmente. Cada PE é de certa forma uma unidade passiva não tomando parte no processo de leitura e descodificação de cada uma das instruções, sendo portanto geralmente processadores menos complexos, consistindo numa simples unidade funcional acoplada a uma memória local de tamanho variável. Por esta razão a maioria das máquinas explorando este tipo de paralelismo apresentam um número elevado de processadores, sendo por vezes referidas como **computadores massivamente paralelos**[5].

A rede de interligação entre os PE, que é igualmente controlada pela unidade de controlo global, pode diferir de máquina para máquina, desde uma ligação completa entre todos os processadores, à existência de apenas uma ligação parcial, o que tem implicações importantes na comunicação de dados entre processadores. *Array Computers* são no fundo

[3] O que sucede sempre que se passa a executar um conjunto de instruções diferente, necessitando de estágios com funcionalidade diversa.

[4] Supercomputadores são máquinas de elevado desempenho, e também elevado custo !

[5] A designação massivamente paralelo não deve no entanto ser utilizada de forma generalizada para designar esta classe de máquinas, tendo apenas recentemente passado a ser utilizada com maior frequência.

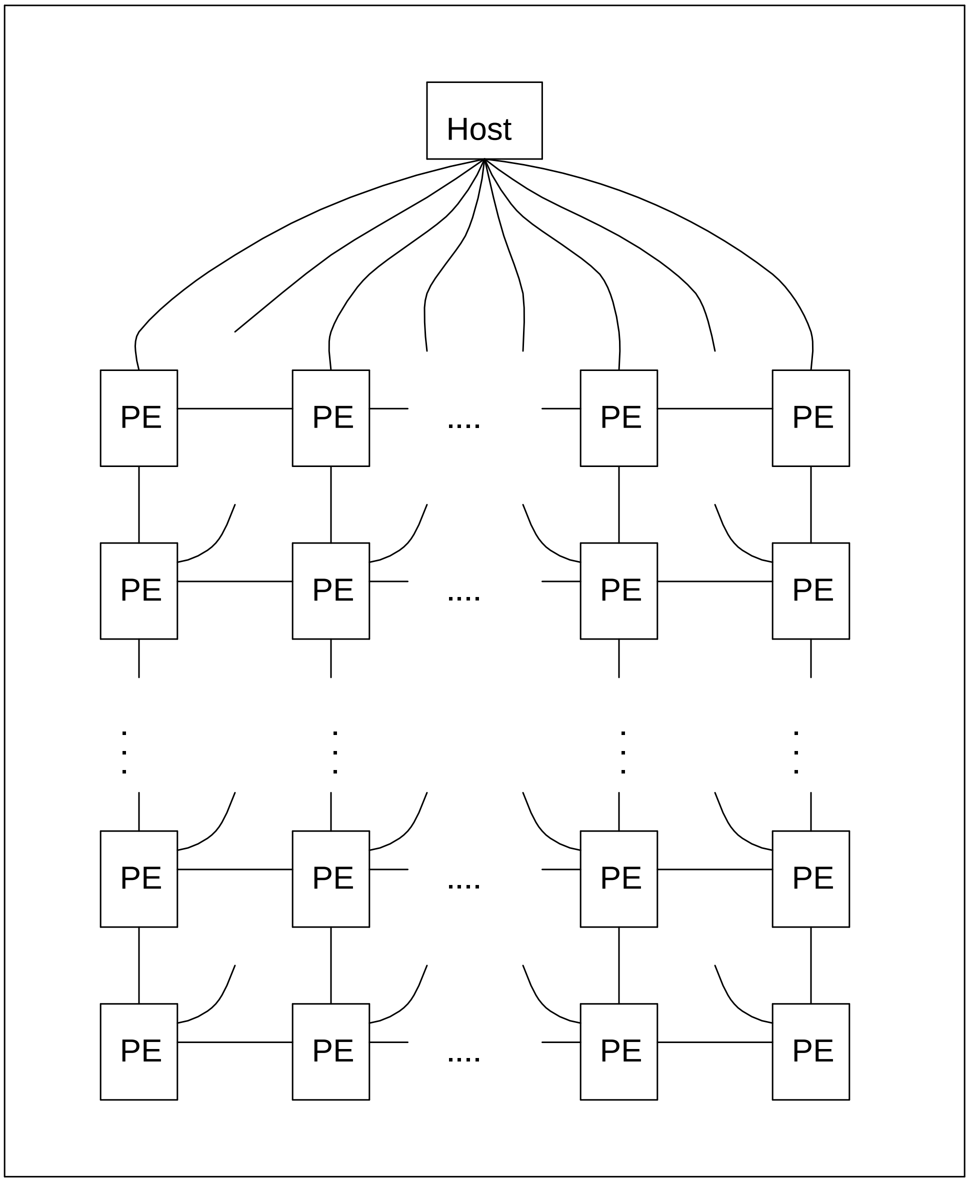

Figura 2.4: Rede bidimensional de processadores

máquinas do tipo SIMD, na medida em que uma única instrução, difundida para todos os PE, é executada sobre vários grupos de dados simultaneamente, uma vez que os dados são lidos localmente na memória de cada PE. Como exemplo de uma arquitectura explorando paralelismo espacial veja-se a figura 2.4 que apresenta uma arquitectura do tipo matricial, ou a arquitectura da **Connection Machine**, na figura 4.1.

A aceleração de execução numa estrutura de *Array Computers* contendo n PE's é teoricamente n. A necessidade de distribuição de um controlo global e a existência de dependência entre dados, faz com que esse valor raramente seja alcançado.

2.1.3 Paralelismo Assíncrono

Os sistemas **Multiprocessadores** são compostos por dois ou mais processadores independentes, de capacidades aproximadas, que partilham o acesso, seja a uma memória comum ou a um conjunto de periféricos de entrada-saída. Cada processador pode ou não ter um banco de memória local e a transferência de dados entre processadores pode ser feita de uma grande variedade de formas, através de um sistema de mensagens associadas a interrupções em canais de leitura-escrita, ou pelo mapeamento da memória total como um único banco de memória. Um único sistema de exploração integrado controla o sistema na sua totalidade, as interacções entre processadores e a sua execução de programas.

A maioria dos Multiprocessadores são máquinas que se podem enquadrar na categoria MIMD, o que necessariamente implica interacção entre os processadores se a memória for partilhada. Se os dados não forem de forma alguma partilhados, a estrutura será então mais correctamente do tipo MSISD ou seja Múltiplos, mas independentes e desacoplados, sistemas SISD. Este tipo de sistemas são vulgarmente designados por multicomputadores e no extremo serão o que se convencionou chamar de redes locais de computadores. A distinção é baseada no grau de cooperação e de partilha de recursos observado na resolução de problemas. Num multiprocessador, um único sistema de exploração controla a entrada de um processador em actividade, a marcação de um outro para um instante posterior, as interacções entre processadores e os seus programas seja ao nível de processo, conjunto de dados, elementos, etc.

A aceleração de execução conseguida com um Multiprocessador baseado em n processadores é teoricamente n. No entanto, tal raramente é possível dado que a partilha de dados entre as várias tarefas leva à necessidade de sincronização entre elas. Isto implica que o processador no qual uma tarefa se executa, pode ser colocado num estado de espera, até que um determinado conjunto de dados fique acessível.

2.1.4 Interligação entre Processadores

Um aspecto que distingue os vários tipos de arquitecturas paralelas é a estrutura de interligação entre os vários processadores. É usual classificar estas estruturas em duas categorias: as baseadas em *bus* e as não baseadas em *bus*.

As estruturas ou redes não baseadas num *bus*, podem ir desde ligações completas, até um conjunto mínimo de ligações dedicadas. Os sistemas completamente ligados são aqueles em que um caminho único e directo existe, em qualquer instante, entre quaisquer dois processadores. Isto pode ser conseguido através da existência de ligações físicas entre todos os processadores[6], ou através da existência de um sistema de ligação dinâmica utilizando um conjunto reduzido de ligações físicas. Poucos exemplos comerciais de máquinas completamente ligadas por ligações físicas existem, e essas utilizam apenas um reduzido número de processadores. Utilizando técnicas de comutação de linhas, existem no entanto várias máquinas, como os *Butterfly* da companhia *BBN (Bolt, Beranek & Newman)* , com desempenho apreciável. Em qualquer caso, em sistemas não baseados em *bus*, a memória pode ser local a cada processador ou global, sendo o primeiro, o único caso realmente interessante na práctica.

As estruturas mais utilizadas em máquinas não baseadas em *bus* são as realizadas a partir de ligações em **rede bidimensional** ou **Hipercubo** [Seitz85] [Silveira88i]. Uma rede bidimensional, como um nome indica, é um matriz em que cada elemento é um processador que está ligado a quatro vizinhos nas direcções dos pontos cardeais. Exemplos destas máquinas são estruturas recentes baseadas no *Transputer* da *Inmos*, ou mais antigas como o *DAP ICL*. Uma estrutura em hipercubo é aquela em que cada um dentre n processadores ocupa um vértice de um cubo de $\log_2 n$ dimensões, estando ligado portanto a outros $\log_2 n$ processadores através de ligações físicas e dispondo da capacidade de se ligar dinâmicamente a qualquer outro processador através de um processo de encaminhamento de mensagens. A razão porque tais estruturas são tão utilizadas e foram tão intensivamente estudadas deve-se às suas características universais. Estas podem ser estabelecidas com precisão e derivadas recursivamente. O estudo teórico das propriedades deste tipo de estruturas bem como de outras muito semelhantes, é um assunto extremamente interessante mas demasiado envolvente para a profundidade com que se pretende aqui abordar o tema [Leigthon]. Na figura 2.5 apresentam-se hipercubos de, respectivamente, uma, duas e três dimensões. Exemplos de máquinas com este tipo de arquitectura são o *iPSC* da *Intel Scientific Computers*, o *NCube/ten* da *NCube*, ou mais recentemente, a *Connec-*

[6]Este tipo de rede, denominada Ligação Física Completa, tem associado grandes custos de espaço e teoricamente implica um número infinito de ligações dado o crescimento exponencial do número destas.

tion Machine da *Thinking Machines Corporation*, entre outros [Wiley87] [Wallich87]. A arquitectura de base da Connection Machine pode ser vista na figura 4.1.

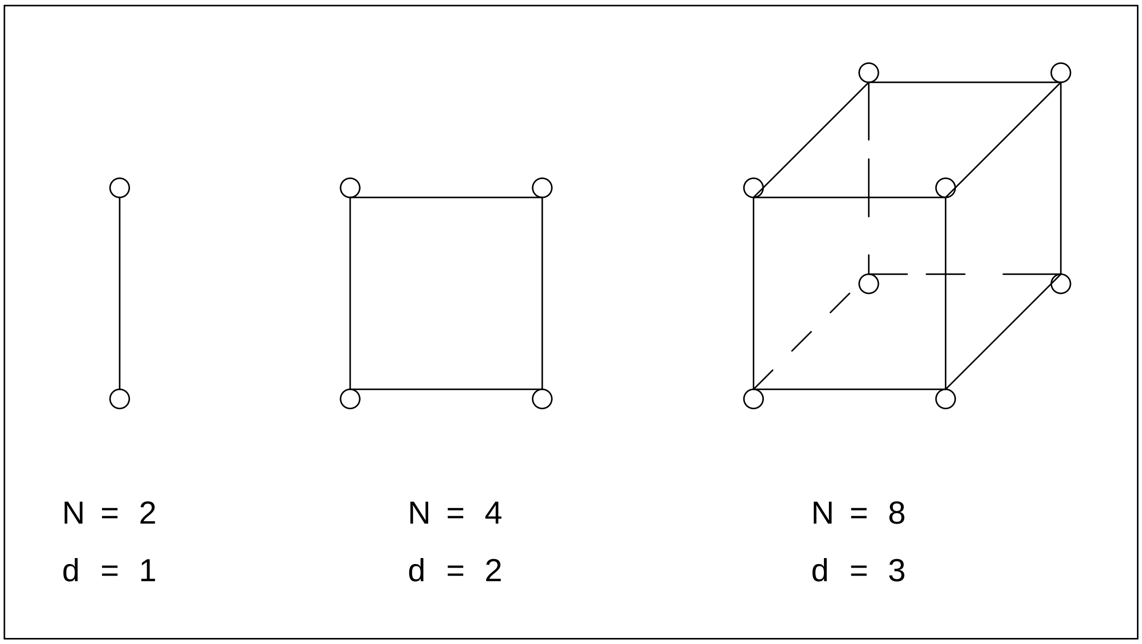

Figura 2.5: Estrutura em Hipercubo de uma, duas e três dimensões

As estruturas baseadas na existência de um *bus*, podem ser subdivididas em sistemas fortemente ligados e sistemas fracamente ligados (*tightly* e *loosely coupled systems*). Sistemas fortemente ligados são aqueles em que a comunicação entre processadores é feita através de uma memória partilhada. A velocidade através da qual dados podem ser trocados é assim, neste tipo de estruturas, da ordem da largura de banda de acesso à memória. A ligação entre os processadores e a memória pode ser efectuada através de uma rede de interligação separada, pelo uso de uma ou mais ligações físicas ou *busses*, ou utilizando memórias multi-porto. A contenção e os conflitos no acesso a memória quando vários processadores tentam ler os mesmos dados concorrentemente, é um factor de degradação e limitativo em termos de expansão do sistema, apesar de existirem formas de minimizar essa contenção e de aumentar a capacidade de processamento de sistemas deste tipo em algumas aplicações. A maioria das máquinas comercialmente disponíveis estão nesta categoria, como o *Alliant FX/8* da *Alliant* [AlliantFX], a série *Balance* da *Sequent Inc*, o *Flex 32* da *Flexible Company*, etc [Wallich86]. Na figura 2.6 apresenta-se um exemplo típico de uma arquitectura baseada em *bus* com memória partilhada.

Sistemas fracamente ligados são aqueles em que a interacção entre processadores é pequena. Nestes sistemas, cada processador tem a sua própria memória local e um conjunto de dispositivos de entrada-saída. Um sistema de transmissão e encaminhamento

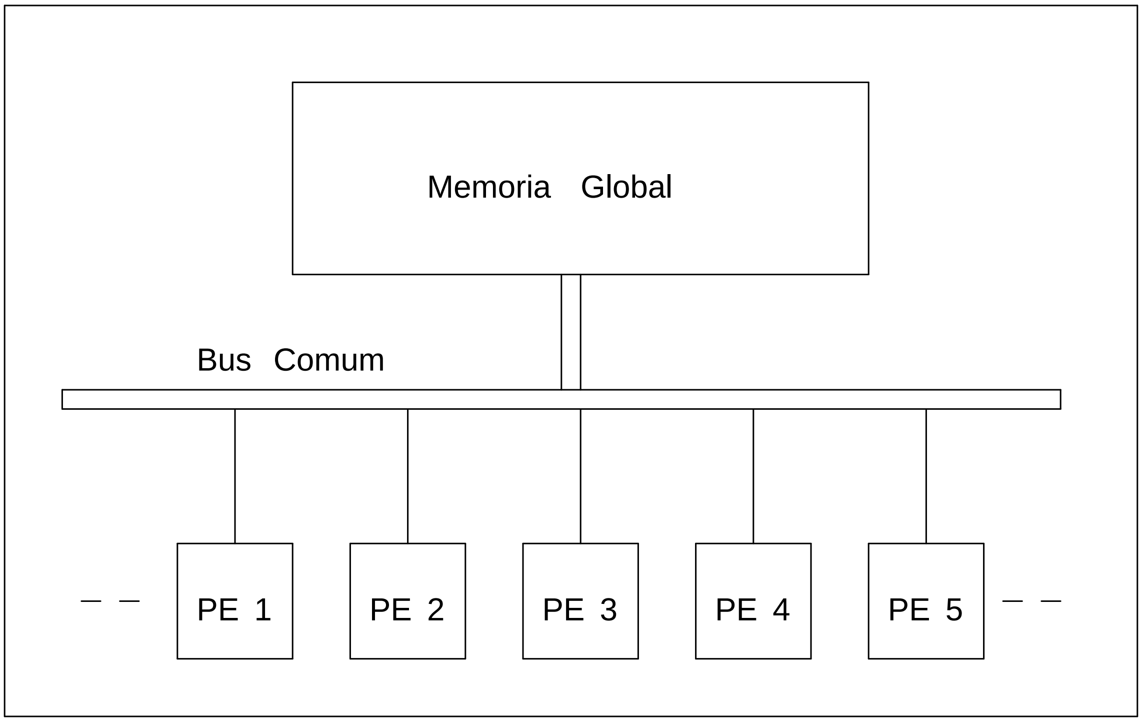

Figura 2.6: Arquitectura baseada em *bus* com memória partilhada

de mensagens zela pela troca de informação entre processos executando em diferentes processadores. O grau de acoplamento e cooperação é pequeno e tais sistemas são eficientes apenas quando as interacções entre tarefas são mínimas, dado que o desempenho se degrada se o número de mensagens no sistema cresce. Naquilo que se poderá chamar a fronteira dos sistemas fracamente ligados, podem-se encontrar as já mencionadas Redes Locais de Computadores, que no entanto certos autores [Wallich86] ainda consideram como arquitecturas paralelas.

2.1.5 Arquitecturas Absolutamente Não Convencionais

As aproximações abordadas nas secções anteriores referem-se a arquitecturas de certa forma convencionais porque baseadas no conceito de fluxo de controlo. Tais arquitecturas no fundo não passam de "meras" extensões ao modelo de Von Neumann, dado que as instruções são executadas de uma forma sequencial e controlada por um contador de programa. Há alguns anos, no entanto, uma estrutura inovadora baseada no fluxo de dados, foi sugerida. Tal estrutura levou ao conceito de **Computadores de Fluxo de Dados** (ou *"data-flow computers"*).

A ideia base de tais estruturas implica a não existência de contadores de programas e a execução de uma instrução sempre que os seus operandos estão disponíveis. O início de uma instrução depende da disponibilidade dos dados, independentemente da localização física de tal instrução na sequência do programa a executar. Isto leva a uma execução não ordenada de instruções o que teoricamente possibilita uma máxima concorrência, limitada apenas pelos recursos disponíveis.

Máquinas deste tipo representam um afastamento ainda mais radical das arquitecturas convencionais do que as arquitecturas paralelas descritas anteriormente. Isso leva a grandes dificuldades no seu desenvolvimento e é a principal razão porque não há máquinas deste tipo comercialmente disponíveis, dado que as construídas até hoje foram-no num ambiente de pesquisa, sendo escassa a informação. O desenvolvimento de tais máquinas é assim extremamente complexo, na medida em que tanto no aspecto arquitectónico como no aspecto de programação elas envolvem conceitos inovadores e não facilmente assimiláveis. Exemplos de aplicações práticas podem no entanto ser encontrados [Hahn85] [Catlin85].

Outros tipos de máquinas, baseados noutros modelos de computação, como as *máquinas de redução*, *os modelos de actores*, ou outros, foram igualmente tentados, embora de novo apenas em ambientes universitários ou de investigação [Dally89].

2.2 Algoritmos para Processamento Paralelo

O interesse crescente no desenvolvimento de máquinas paralelas implica um esforço de pesquisa no sentido de desenvolver ferramentas capazes de eficazmente aproveitarem as potencialidades oferecidas por tais máquinas.

Desde cedo se tornou claro que as técnicas em uso, concebidas para máquinas uniprocessador, testadas e melhoradas através de anos de uso e estudo, não eram, na generalidade dos casos, eficientes para computadores paralelos. Tornou-se então necessário o desenvolvimento de novos algoritmos, especificamente destinados a essas arquitecturas e capazes de minimizarem o custo atribuído à resolução de problemas computacionalmente muito intensivos.

Tal desenvolvimento implica em geral um esforço de exploração de paralelismo em todas as fases de resolução de um problema e requer um conhecimento de certa forma profundo da categoria arquitectónica da máquina à qual o programa se destina.

Existem duas formas básicas pelas quais um algoritmo pode ser observado no sentido de se explorar o paralelismo nele existente: ao nível da sua funcionalidade ou ao nível das estruturas de dados sobre as quais actua. A primeira possibilidade é geralmente designada por **Paralelismo Funcional**, enquanto que a segunda se denomina **Paralelismo de Dados**[7].

Paralelismo Funcional equivale a subdividir um algoritmo num conjunto de tarefas básicas que podem ser efectuadas em paralelo. Tais tarefas podem ser alocadas a processadores diferentes, tirando-se assim partido do paralelismo inerente ao próprio algoritmo. O procedimento utilizado para dividir um algoritmo em unidades funcionais independentes começa por identificar conjuntos de tranformações efectuadas sobre os dados do problema. Tais grupos de operações devem ser fechados, no sentido em que uma vez entrado um conjunto de dados, o mesmo pode ser processado inteiramente apenas por operações dentro do mesmo grupo e de alguma forma, um resultado estará válido no final dessas operações.

A quase totalidade dos problemas computacionalmente onerosos e para os quais urge encontrar formas de aceleração, contém uma característica essencial em comum. São constituídos por uma sequência extremamente grande, ou um ciclo efectuado inúmeras vezes, de operações elementares. Tal leva a que, de forma simples, se possam definir nessa série, ou ciclo de operações, agrupamentos com fronteiras bem definidas e geralmente com alguma funcionalidade lógica associada. Tais grupos de operações podem então ser colocados em sequência numa espécie de *pipeline* algorítmica, de forma a que os dados saídos de um grupo constituam dados de entrada do grupo seguinte.

Partições ao nível funcional conduzem assim a uma *pipeline* em que cada estágio tem associada uma determinada transformação dos dados que recebe. Tais partições são assim inerentemente síncronas e em geral bastante independentes do tipo de arquitectura para as quais são concebidas. De facto, a fronteira de cada unidade funcional é definida por um conjunto de entradas e saídas de tal forma que apenas esses dados têm de ser comunicados de ou para a unidade.

Paralelismo de Dados equivale a considerar os dados de um problema como um conjunto que pode ser subdividido de tal forma que o mesmo ou diferentes conjuntos de operações podem ser efectuados simultaneamente sobre dados diferentes. Em princípio, tal esquema parece ser perfeitamente semelhante ao descrito anteriormente, mas de facto a ênfase, neste caso, é dada ao agrupamento de dados de certa forma relacionados, em conjuntos que podem ser tratados como estruturas únicas a um nível mais elevado da

[7] As designações Partição Funcional e Partição de Dados são igualmente comuns. Ambas serão utilizadas indistintamente no que se segue.

hierarquia de dados. A partição dos dados de um algoritmo pode ser efectuada dinâmica ou estaticamente. Desta forma é possível explorar paralelismo temporal e espacial ao nível do algoritmo.

Partições de dados efectuadas estaticamente são sempre mais simples de implementar, mas podem conduzir a um desequilíbrio na taxa de ocupação de cada um dos processadores. De facto, se os dados são agrupados estaticamente, pode suceder que um ou mais grupos de dados, dependendo dos estados anteriores do algoritmo, nunca seja actuado, o que no caso de ter sido alocado estáticamente a um processador coloca-lo-á num estado de espera infinitamente longo, o que representa um desperdício de recursos.

A questão que acaba de se levantar pode ser igualmente posta no caso de uma partição funcional. Enunciando o problema de maneira mais geral, a pergunta que se põe é a de como se devem alocar as partições (funções ou dados) aos processadores existentes, de forma a maximizar a eficiência obtida. Infelizmente não existe uma resposta geral para este como para outro tipo de questões fundamentais, devendo a mesma ser procurada individualmente ao nível de cada algoritmo que se pretende implementar e da arquitectura a que se destina. Determinado tipo de algoritmos, em geral baseados numa estrutura bem definida e balanceada, são indicados para um esquema de alocação estático. Outros, de estrutura física ou computacional mais irregular, são mais indicados para esquemas de alocação dinâmicos. O processo de alocação, por sua vez, pode variar desde uma alocação aleatória, em geral extremamente rápida, mas ignorando qualquer estrutura do problema, a uma alocação por traço[8], seguindo a sequência de interacção das partições, ou outro tipo, considerado mais eficiente para o problema em questão.

Tal como se viu para o caso das arquitecturas de processamento paralelo em que a adição de processadores aumenta o número de graus de liberdade e conduz a categorias arquitectónicas distintas, tal também sucede no aspecto algorítmico. Claramente, a arquitectura da máquina tem uma influência fundamental no desempenho de um algoritmo paralelo. As opções tomadas no desenho da máquina implicam restrições fundamentais a impôr a qualquer algoritmo que aí se implemente.

A verdadeira questão por detrás de todos estes problemas, muitas vezes mal compreendida ou desprezada, tem no entanto a ver com o problema da comunicação entre processadores. O problema da comunicação e as necessidades de partilha de dados são de facto a questão fundamental na escolha de uma dada arquitectura ou de uma dada aproximação para uma implementação paralela.

[8]Uma alocação por traço processa-se colocando um conjunto de elementos numa lista ordenada e percorrendo então essa lista alocando-os aos processadores disponíveis.

É vulgar definir um parâmetro de grande importância num algoritmo paralelo, como sendo a razão entre as necessidades computacionais e de comunicação do algoritmo. Tal parâmetro, designado por **Granularidade do Algoritmo**, mede, em geral de forma imprecisa, se um algoritmo é ou não localizado em relação ao tipo de partição explorada.

Para partições funcionais podemos ver que **algoritmos de grão fino** *("fine-grain parallelism")* são aqueles em que cada unidade funcional corresponde a um conjunto reduzido de operações[9], ou apenas uma operação, enquanto que **algoritmos de grão grosso** *("coarse-grain parallelism")* correspondem em geral a poucas unidades funcionais, de grande complexidade. Partições funcionais de grão grosso têm a grande desvantagem de que em geral não são expansíveis com o número de processadores, não se obtendo aumento de desempenho quando se aumenta o número de unidades computacionais, a menos que se dupliquem as unidades básicas e se use uma rede de interligação muito versátil e eficiente, situação em que teremos uma partição mista.

Para partições de dados, algoritmos de grão fino baseiam a sua estrutura no tratamento de conjuntos de dados extremamente reduzidos, o que pode implicar troca de informação a uma razão elevada. Ao contrário, algoritmos de grão grosso, levam em geral a grupos de dados de consideráveis dimensões, e de certa forma fortemente relacionados.

O problema de transferência de dados entre processadores, ou entre unidades funcionais, é claramente condicionado pelas possibilidades arquitectónicas da máquina à qual a implementação se destina. Em arquitecturas do tipo centralizado, em que os dados são partilhados, ou naquelas em que canais dedicados existem entre os vários processadores ou processadores e memória, faz sentido utilizar algoritmos de grão fino, dado que se torna relativamente barato transferir dados. Se no entanto na transferência desses dados, todos os processadores utilizarem um meio único (caso das arquitecturas baseadas em *bus*), então um excesso de comunicação provoca contenção no acesso a esse meio, o que degrada fortemente o desempenho de um algoritmo.

As questões relacionadas com o desenvolvimento de algoritmos para aplicações em máquinas paralelas são assim extremamente complexas e, em geral, não é possível encontrar soluções únicas e universais. As formas de exploração do paralelismo, o nível de detalhe a que esse paralelismo é explorado, e logo as necessidades em termos de comunicação desse algoritmo, estão fortemente interligadas com a arquitectura da máquina à qual a implementação se destina.

[9] De forma muito imprecisa, um algoritmo *fine-grain* pode ser considerado como uma forma de *"data-flow"*.

2.3 Ferramentas de CAD em Arquitecturas Paralelas

As ferramentas utilizadas em processos de CAD (Projecto Assistido por Computador) ou CAD/CAM (Projecto e Fabrico Assistidos por Computador) pertencem a um conjunto de programas de grande utilização, e computacionalmente bastante envolventes[10] [Schendel84].

As ferramentas de CAD são destinadas a aumentar a fiabilidade de um projecto, a maximizar o seu desempenho, facilitar o seu desenvolvimento, e diminuir o tempo de retorno desde que tais melhoramentos são gerados até à fase de produção. Exemplos de ferramentas deste tipo são: Sistemas de Disposição Automática de Componentes numa Carta de Circuito Impresso, Desenho de Circuitos Integrados, Colocação de dispositivos em Circuitos Integrados e sua interligação, Simulação de Circuitos, etc.

Um esforço de investigação tem sido desenvolvido para obter melhorias no desempenho de praticamente todas estas ferramentas, passando pela sua implementação em arquitecturas de processamento paralelo. Problemas como Disposição de componentes, Ligação entre dispositivos, etc, têm uma estrutura física associada que lhes confere um elevado grau de paralelismo [Rutenbar88]. Noutras ferramentas, outros tipos de paralelismo são explorados. Variados exemplos de implementações de ferramentas de CAD em máquinas paralelas foram já apresentados, sendo uma percentagem elevada dos mesmos baseados em Processadores especializados ou dedicados [Arnold85]. Uma parcela significativa destes aceleradores, são implementações de simuladores ao nível lógico [Uno85] [Sasaki83] [Denneau82], embora um conjunto de simuladores ao nível eléctrico [Lewis85], ou ao nível do interruptor [Sasaki86] tenham sido apresentados.

A simulação ao nível lógico é, tradicionalmente, a utilizada para simular sistemas de elevada dimensão, devido ao seu reduzido custo por dispositivo. Desta forma foi uma das primeiras ferramentas a ser alvo de pesquisa no sentido de se acelerar a sua execução, à medida que a complexidade e dimensão dos circuitos crescia. A Simulação lógica tem no entanto um conjunto de características específicas que a tornam extremamente interessante do ponto de vista de paralelização. Por um lado tais simuladores trabalham sobre estruturas unidireccionais, sendo portanto fácil estabelecer relações de interacção directa entre parcelas do circuito. Por outro lado trabalham em geral sobre uma escala de tempos discreta, dado que admitem modelos de atraso mínimo para os componentes.

[10] Outros exemplos a ter em conta são: previsão metereológica, dinâmica de fluidos, análise sísmica, etc.

Por razões que se tornarão mais claras adiante interessa aqui debater um pouco a questão do paralelismo explorado nos simuladores do tipo lógico. As implementações de simuladores a este nível seguem em geral uma de duas linhas estratégicas. Um número reduzido [Denneau82], baseado em arquitecturas que posteriormente se viriam a denominar de Massivamente Paralelas, assenta numa aproximação de “força bruta”, em que para cada instante, se resolve simultâneamente todo, ou uma grande parte do circuito. Esta avaliação provoca naturalmente a mudança de estado lógico de alguns nós internos, o que será tido em conta na avaliação seguinte. Tal aproximação, para circuitos de grande dimensão, é claramente apenas apropriada para máquinas com elevado número de processadores. Note-se a importância da discretização da escala de tempos baseada em parâmetros do modelo dos dispositivos e que permite seguir a evolução do circuito calculando-o apenas num conjunto de pontos. Tal não sucede por exemplo ao nível eléctrico em que não existe à priori uma escala mínima de tempos[11] [Jacob85].

Um outro tipo de implementações baseia-se no conceito de simulação baseada em acontecimentos (*event-driven simulation*) [Sasaki83]. Um algoritmo de simulação diz-se controlado por acontecimentos, quando o controlo da evolução da simulação depende da marcação e resolução de acontecimentos associados a alterações de estado no circuito. Neste tipo de implementações, o paralelismo é de novo aproveitado em cada instante de tempo, resolvendo-se simultaneamente todos os blocos lógicos cujo estado foi marcado para alteração nesse instante. Implementações deste tipo são apropriadas para arquitecturas com um número mais reduzido de processadores, podendo ser utilizadas em máquinas com um número superior de processadores, no caso de circuitos de muito grande dimensão.

Os simuladores ao nível do interruptor são menos facilmente paralelizáveis que os do nível lógico. Em particular, tal como ao nível eléctrico, não existe o conceito de atraso mínimo pelo que o eixo dos tempos é tratado como contínuo. Os simuladores *switch-level* são baseados no conceito de grupo ou conjunto de nós fortemente ligados, sendo à partida propícios para uma partição de dados. No entanto, tal como os simuladores lógicos, implementações mistas, explorando paralelismo de dados e funcional, podem ser encontradas e obtendo resultados muito satisfatórios.

Finalmente, também ao nível eléctrico várias implementações foram já feitas [Sadayappan] [Gallivan] [Webber87], dado que, como se viu, é de entre os níveis normalmente usados o mais oneroso. Infelizmente, é igualmente o de maior dificuldade devido a um conjunto de características que adiante serão expostas.

[11] Exceptuando, como é evidente, a imposta pela precisão finita de um computador.

Capítulo 3

Contexto do Trabalho

Neste capítulo, pretende-se introduzir, sob uma perspectiva de certa forma generalizada, o contexto em que este trabalho foi realizado. Para tal começa por se descrever o ambiente global em que o mesmo foi desenvolvido, e a forma como nele se enquadra. Em seguida aborda-se, de forma mais pormenorizada, o problema particular da simulação eléctrica de circuitos nos seus aspectos gerais, focando a atenção no paralelismo inerente ao problema.

Acentua-se em seguida o carácter mais contextual dos temas abordados neste capítulo, ao apresentar-se o ambiente concreto de trabalho, introduzindo-se o simulador **CINNAMON**, discutindo-se a metodologia que lhe está inerente e o desenvolvimento de que tem sido alvo. Faz-se um estudo prévio do paralelismo de um tal tipo de algoritmos, estabelecendo-se paralelos com outras implementações. Termina-se com uma breve descrição da arquitectura da máquina que serviu de suporte às implementações descritas no capítulo seguinte.

3.1 Projecto Esprit-1058

Este trabalho foi desenvolvido no âmbito de um programa de cooperação europeu, o **Projecto ESPRIT 1058** - *Knowledge Based Design Assistant for Modular VLSI Design* [1058Proposal], financiado em parte pela Comissão da Comunidade Económica Europeia. Os projectos ESPRIT são projectos destinados ao sector das tecnologias de informação, seja sob a forma de processamento ou de transmissão dessa informação, que aspiram a um desenvolvimento mais eficaz e coeso da tecnologia e conhecimento no campo, dentro dos limites da Europa Comunitária. São participantes neste projecto, além do Instituto de Engenharia de Sistemas e Computadores (INESC), a companhia belga de ferramentas de CAD *Silvar-Lisco*, a Philips holandesa através do seu centro de CAD de Eindhoven e o Centro de Microeléctrónica belga *IMEC*.

O projecto 1058 tem por objectivo desenvolver um conjunto coerente de programas, flexíveis, portáteis e fáceis de utilizar, para o projecto de circuitos integrados de grande porte. Está directamente relacionado com o tratamento dos dados relacionados com as várias fases do projecto, e a sua interacção de forma flexível e eficiente. Estes programas suportam técnicas modernas de concepção e verificação de circuitos, num ambiente de projecto assistido por computador. Tais programas pretendem-se integrados sobre uma plataforma de dados comum, por forma a facilitar a transmissão e livre utilização dos mesmos pelas várias ferramentas utilizadas durante a fase de desenvolvimento de um projecto. O projecto tem ligações com outros projectos dentro do programa ESPRIT, podendo considerar-se como um elo de uma cadeia mais completa que pretende cobrir as necessidades de um ambiente de projecto de circuitos integrados actual.

O objectivo da participação do INESC neste projecto, através do envolvimento do projecto Simulação, consiste no desenvolvimento de uma ferramenta específica de verificação e análise de projecto. Para tal propõe-se, como se descreverá adiante, o estudo e desenvolvimento de novos algoritmos de simulação de circuitos, com base numa estratégia inovadora para a resolução do problema [Neto86i] [Neto86ii]. Tal passa pelo desenvolvimento de um simulador de circuitos ao nível eléctrico, e estudo de técnicas de aceleração do processo de simulação, seja por recurso a metodologias multi-nível ou pela implementação do algoritmo de base em arquitecturas paralelas.

Este trabalho surge assim como agregado ao conjunto de tarefas que o INESC pretende desenvolver no contexto do projecto ESPRIT-1058. Pretendem-se estudar metodologias e processos que conduzam a uma simulação mais rápida, mantendo a precisão de um modelo de nível eléctrico. Recorre-se para tal ao conceito de simulação paralela de circuitos.

3.2 Simulação Eléctrica de Circuitos

A simulação eléctrica de circuitos é hoje em dia uma ferramenta indispensável no projecto de um circuito integrado. Dada a dimensão dos circuitos que as tecnologias actuais permitem fabricar, devido à elevada escala de integração acessível actualmente, a simulação eléctrica é responsável por uma elevada fatia do custo computacional do desenvolvimento de um projecto. Trata-se de um tópico de grande interesse, atendendo exactamente ao seu uso intensivo, que se traduz num consumo de milhares de horas de computador por ano. Ao fabricar um circuito, um projectista está na verdade a confiar elevadas quantias monetárias à verificação realizada por simulação, pelo que esta deve ser extremamente robusta e precisa.

A simulação eléctrica permite analisar e verificar um circuito a um nível de elevado detalhe, possibilitando porventura uma fabricação correcta do primeiro silício[1]. Uma simulação ao nível eléctrico pode ser efectuada para determinar as condições de funcionamento em repouso de um circuito (análise *DC*), a sua resposta em frequência (análise *AC*), a resposta a transitórios nas suas entradas (análise *Transiente*), ou um conjunto de características que permitem medir, de uma ou outra forma, o desempenho do circuito.

Dentre as possíveis análises efectuadas ao nível eléctrico, a análise transitória no domínio do tempo é não só, claramente, a mais utilizada e habitual, mas igualmente aquela que tem maior custo computacional associado. Em contrapartida, fornece informação extremamente precisa sobre o estado interno de um circuito, o seu desempenho e as suas características essenciais.

Existem diversas formas pelas quais a análise transitória pode ser realizada, dependendo da metodologia seguida em cada um dos passos da análise. No entanto, qualquer algoritmo pode ser dividido num conjunto de fases cuja implementação dependerá dos algoritmos utilizados. Tais fases podem em geral ser descritas como:

- Leitura dos dados do circuito a simular. Tal inclui não só a leitura da topologia do circuito, como informação adicional sobre o tipo de análise a efectuar, dados sobre os modelos dos dispositivos e dados relacionados com a duração e o controlo da simulação.

- Formulação das equações que descrevem o circuito com base em modelos internos, e dados pertinentes a cada problema em particular.

- Resolução das equações do circuito, com base num algoritmo que envolve necessariamente manipulação algébrica de algum tipo, e determinação da solução pretendida.

- Geração de documentação sobre a análise efectuada e os resultados obtidos.

A leitura dos dados do circuito envolve a leitura da topologia do mesmo e do valor dos parâmetros de cada elemento. Ao nível eléctrico, os elementos que representam um circuito são Resistências, Condensadores, Fontes de Tensão e Corrente, Bobines, Fontes

[1] Os circuitos MOS são construídos sobre cristais de silício. Designa-se por fabricação em primeiro silício a um projecto correcto na primeira integração.

Controladas, Linhas de Transmissão, Transformadores e elementos semicondutores, como Díodos, Transistores Bipolares e MOS, etc. Em geral, na maioria dos simuladores estes dados estão contidos num ficheiro que contém igualmente a informação necessária ao controlo da simulação, como o tempo de simulação desejado, condições inciais nos nós e outros dados de interesse. Esta primeira fase do processo é no entanto, de certa forma, trivial e não nos interessa especialmente neste contexto. No entanto a existência de um analisador sintático *(parser)* eficiente e capaz de proporcionar uma interface agradável e simples, é de extrema importância. Isto porque se a interface é rígida e complicada, o projectista tem dificuldade em descrever o seu circuito nela, e tende a abandonar o simulador como ferramenta; por outro lado, se o *(parser)* não é eficiente, pode tornar-se dispendiosa a sua utilização no caso de se simularem circuitos de grandes dimensões.

A formulação das equações e a sua resolução, correspondentes às fases seguintes de um processo de simulação ao nível eléctrico, são as fases de maior importância. Não só porque são o que distingue o nível eléctrico dos outros níveis de representação, mas igualmente e sobretudo, porque são as fases de maior computação de todo o processo.

A última fase, de geração de documentação sobre o processo de simulação, é sem dúvida importante, pois é a partir da documentação gerada que os resultados da simulação vão ser analisados. Tal documentação, seja sob forma gráfica ou escrita, deve ser assim suficientemente detalhada de molde a fornecer a informação desejada, mas clara e sucinta de forma a tornar possível o seu entendimento.

Tanto as fases de inicialização como de finalização do processo de simulação são assim claramente importantes e de certa forma tão essenciais para um programa de simulação como as fases numéricas do mesmo. No entanto, qualquer delas é inerentemente sequencial, o que as retira dos objectivos deste trabalho. Assim, a aceleração do processo de simulação, em última medida, reside na aceleração da parte numérica dos algoritmos, nomeadamente a formulação e resolução das equações do circuito.

3.2.1 Formulação das Equações

O comportamento eléctrico de um circuito no domínio do tempo, pode ser descrito com elevada precisão por um conjunto de equações envolvendo a tensão nos nós interiores do circuito, e a corrente, carga ou fluxo nos dispositivos. A formulação das equações do circuito é feita com base nas Leis de Kirchoff e nas equações constitutivas dos elementos (assumidos como representáveis por parâmetros concentrados). As leis de Kirchof traduzem um conjunto de propriedades associadas a um circuito e são inteiramente determinadas a partir da sua topologia. As equações constitutivas são representações matemáticas das relações entre as variáveis mencionadas acima, para o elemento em questão, e que em geral dependem da estrutura física do dispositivo.

As leis de Kirchoff e as equações constitutivas caracterizam de forma completa o comportamento de um circuito. A escrita e resolução das equações daí resultantes é suficiente para determinar todas as variáveis de interesse de um circuito. O método mais utilizado em simulação eléctrica de circuitos consiste na formulação das equações através do **método nodal modificado** [Chua75] [Vlach83]. A utilização deste método consiste na escrita das equações que representam a verificação da **Lei de Kirchoff dos Nós**. Em seguida as equações constitutivas dos elementos são usadas para expressar as correntes nesses elementos em termos das tensões. Este processo necessita de ser extendido no caso de elementos em que a tensão não seja a variável de controlo, o que é facilmente realizado. Finalmente a **Lei de Kirchoff das Tensões** é aplicada para expressar as tensões nos elementos em termos da tensões dos nós. O resultado é um conjunto de equações, geralmente não lineares, cujas incógnitas são as tensões nos nós.

Em particular, se o circuito contiver apenas elementos cuja corrente é controlada pela tensão chega-se a um sistema de equações, não linear do tipo:

$$\frac{\partial}{\partial t} q(v(t), u(t)) = i(v(t), u(t)) \tag{3.1}$$

em que $v(t)$ é o vector das incógnitas (tensões nos nós), $u(t)$ o vector das fontes do circuito, $q()$ o vector da soma das cargas capacitivas em cada nó, $i()$ o vector da soma das correntes resistivas em cada nó. Note-se que as equações anteriores aplicam-se ao caso de circuitos apenas com condensadores, resistências e fontes satisfazendo determinadas condições topológicas. A metodologia pode no entanto ser facilmente extendida ao caso geral [White86]. Este é o tipo de aproximação seguido na grande maioria dos simuladores em uso actualmente [Nagel75] [Newton84] [Lumsdaine88].

Uma metodologia um pouco diversa consiste em definir

$$C(v(t), u(t)) = \frac{\partial}{\partial v} q(v(t), u(t)) \tag{3.2}$$

como sendo a matriz das capacidades do circuito, e

$$f(v(t), u(t)) = i(v(t), u(t)) - \frac{\partial}{\partial u} q(v(t), u(t)) \frac{d}{dt} u(t) \tag{3.3}$$

como a matriz dos elementos resistivos, conduzindo ao sistema de equações:

$$C(v(t), u(t))\ \frac{\partial}{\partial t} v(t) = f(v(t), u(t)) \qquad\qquad v(0) = v_0 \tag{3.4}$$

Embora este segundo método tenha algumas desvantagens, em particular porque conduz a situações em que é violado o pricípio de conservação da carga, é possível minimizar esses efeitos e conseguir resultados correctos [Weeks73], pelo que pode normalmente ser utilizado.

Uma vez construído o sistema não linear acima é necessário resolvê-lo para obter a função $v(t)$ das tensões nos nós, num dado intervalo $t \in [0, T]$ e dada uma condição inicial $v(0) = v_0$. Independentemente de se usar a formulação da equação 3.1 ou da equação 3.4, tal consiste na resolução de um **sistema de equações diferenciais ordinárias não linear**, num dado intervalo de tempo, dada uma condição inicial nos valores das tensões[2]. Em qualquer dos casos é possível demonstrar matematicamente um conjunto de propriedades associadas com as equações anteriores e os métodos utilizados para a sua resolução. Tal por si é um assunto de demasiada extensão para a análise que se pretende fazer aqui, sugerindo-se como referências [Chua75] ou [White86].

3.2.2 Metodologias de Resolução do Problema

Uma vez obtido o sistema de equações que descreve o sistema, é necessário utilizar um conjunto de técnicas de cálculo e álgebra para determinar a solução desse sistema. Existem diversos métodos para resolução do sistema de equações diferenciais ordinário, não linear resultante. No que se segue serão descritas as metodologias usuais de resolução do problema, no seu aspecto conceptual, omitindo os detalhes que podem ser vistos nas referências.

A solução comum de resolução do sistema anterior é vulgarmente referida como **Métodos Directos** sendo utilizada por exemplo no simulador mais vulgar actualmente, SPICE [Nagel75]. Consiste na execução de três passos essenciais:

- Utilização de métodos de integração implícitos para converter o sistema de equações diferenciais num sistema de equações não linear algébrico.

[2]Esta é geralmente obtida previamente através de uma análise DC ou por escolha apropriada de valores de tensão.

- Utilização de um algoritmo de resolução de sistemas não lineares, como o Método de Newton-Raphson, para converter o sistema de equações não lineares num sistema linear.

- Resolução do sistema linear através do Método de Gauss ou de decomposição LU.

O primeiro passo descrito consiste basicamente em discretizar o intervalo de integração num número de pontos no tempo em que uma solução numérica é calculada e utilizar um passo de integração, dependente da discretização, para avançar de um ponto a outro na escala de tempos. A solução encontrada em cada ponto no tempo é obtida utilizando informação anteriormente calculada e a dinâmica do sistema. Para tal são utilizadas **Fórmulas Lineares Multi-Passo**, em geral implícitas, que discretizam o eixo dos tempos [Trefethen].

Considerando a equação diferencial ordinária de primeira ordem:

$$\frac{d}{dt}u(t) = f(u(t), t) \qquad u(0) = u_0 \qquad t \in [0, T] \tag{3.5}$$

uma fórmula multi-passo de passo s aplicada ao problema anterior tem a forma geral:

$$\sum_{i=0}^{s} \alpha_i v^{n+1-i} = h \sum_{i=0}^{s} \beta_i f^{n+1-s} \tag{3.6}$$

em que h é o passo temporal resultante da discretização, v^n entende-se como o valor estimado da solução no instante de tempo $t = t_n = hn$ (o n em v^n é um sobrescrito e não uma potência), $f^n = f(v^n, t_n)$ é a dinâmica da equação no valor estimado e no ponto correspondente e os parâmetros α_i e β_i são escolhidos para maximizar a precisão da fórmula dentro de certos limites de estabilidade.

Como se depreende da fórmula geral anterior (equação 3.6), a solução da equação diferencial é determinada ponto a ponto, sendo o valor da função num dado instante de tempo determinado a partir dos valores em instantes anteriores e da dinâmica da equação diferencial. Um exemplo, extremamente utilizado em simulação de circuitos, é o Método Trapezoidal:

$$v^{n+1} = v^n + \frac{h}{2}(f^{n+1} + f^n) \tag{3.7}$$

e outro são as chamadas fórmulas de Diferenciação Regressiva ou Método de Gear, que para passo $s = 2$ tomam a forma:

$$v^{n+1} = -\frac{4}{3}v^n + \frac{1}{3}v^{n-1} + \frac{2}{3}h\ f^{n+1} \tag{3.8}$$

De notar que se na equação 3.6 se tiver $\beta_0 \neq 0$ temos o que se chama um **Método Implícito**, caso contrário, $\beta_0 = 0$, temos um **Método Explícito**. Como se pode depreender, a utilização de um método explícito torna o problema bastante mais simples, ao possibilitar o desacoplamento total das equações. Infelizmente há considerações de estabilidade a ter em conta, tendo-se demonstrado que os métodos implícitos são mais estáveis que os explícitos. Estes requerem assim a utilização de passos de integração menores, de forma a serem estáveis e convergirem para a solução correcta. Esta é razão porque em geral os simuladores utilizados usam fórmulas implícitas como as apresentadas acima. A utilização de métodos implícitos implica no entanto que as equações do sistema resultante não são desacopladas, o que torna a sua resolução bastante mais onerosa.

Uma vez utilizada uma fórmula multipasso para transformar o sistema de equações diferenciais num sistema não linear, este é resolvido com uma qualquer variante do método de Newton, o que leva a um sistema linear que é necessário resolver iterativamente até à convergência [Wing80]. Considerando o sistema de equações na forma:

$$F(v(t)) = 0 \tag{3.9}$$

e aplicando o **método de Newton-Raphson**, tem-se:

$$v^{k+1}(t) = v^k - J(F(v^k), v^k)\ \ F(v^k) \tag{3.10}$$

em que v^k é o vector das tensões nos nós na iteração k, $F(v^k)$ é o resíduo da função nessa iteração, e $J(F(v^k), v^k)$ é o jacobiano da função calculado no ponto v^k.

Em cada iteração deste método, torna-se necessário resolver um sistema algébrico linear de equações, o que é feito por decomposição LU ou eliminação gaussiana. Na prática os sistemas resultantes de circuitos típicos, são em geral bastantes esparsos (muitos elementos nulos) pelo que existem técnicas específicas que permitem a resolução deste tipo de problemas de forma muito mais eficiente [Duff86] [George81].

Em termos gerais, o algoritmo de controlo da simulação, baseado nos métodos directos, pode ser descrito em pseudo-código da forma seguinte:

Algoritmo 3.1 (Simulação Eléctrica utilizando Métodos Directos)

```
simulação()
{
    t = 0 ;
    formulação das equações do circuito ;
    determinação do passo de integração apropriado ;
    discretização do eixo dos tempos por aplicação de uma fórmula multi-
      passo ;
    enquanto( t < TempoMaximo) {
        enquanto( não se verificar convergência) {
            linearização das equações pelo método de Newton ;
            resolução do sistema algébrico resultante ;
            arquivação de resultados ;
            utilização de um critério de redução do passo caso não haja
              convergência ;
        }
        t = t + passo de integração ;
    }
}
```

Um método alternativo de resolução das equações do circuito envolve o uso de **Relaxação** [Newton84] [White86]. Esta pode ser aplicada a vários níveis ao problema anterior. O mais simples, denominado **Relaxação Linear** substitui a resolução do sistema linear descrito pela equação 3.10 por um método iterativo como Gauss-Seidel ou Gauss-Jacobi [Varga62]. No entanto, é geralmente mais eficiente utilizar relaxação numa fase anterior do problema, para aproveitar determinadas características dos circuitos que os métodos anteriores não capturam. Tal é obtido com **Relaxação Não Linear** que corresponde a resolver o sistema não linear descrito pela equação 3.9 em cada ponto no tempo, iterativamente, desacoplando as equações e resolvendo-as por exemplo pelo método de Newton. Em particular, neste caso, prova-se que as características de convergência quadrática do algoritmo de Newton não se perdem se entre cada relaxação apenas se fizer uma iteração do método de Newton. Este método de relaxação tem aspectos interessantes a considerar: em primeiro lugar o desacoplamento das equações permite utilizar passos de integração diferentes para cada uma delas, dependendo da dinâmica local do circuito, aproveitando assim a latência do mesmo. Em segundo lugar não se torna agora necessário resolver um sistema completo, mas apenas várias equações escalares.

Um terceiro método usualmente aplicado, denomina-se **Relaxação de Ondas** [Newton84] [White86] [Dumlugol86]. Corresponde a aplicar um processo iterativo directamente ao sistema de equações diferenciais 3.5 o que implica que as variáveis do circuito não são agora as tensões nos nós em cada ponto no tempo, mas a onda que representa a tensão num dado intervalo de tempo [White85ii]. Tais técnicas envolvem em geral a partição do circuito num conjunto de sub-circuitos, de forma a acelerar a convergência das formas de onda dentro de cada sub-circuito. Tais partições são em geral feitas estaticamente, o que por vezes é extremamente ineficaz. Por outro lado possibilita a utilização de passos de integração diferentes nos vários sub-circuitos, o que pode tornar a simulação bastante mais rápida [Saleh87].

Outro tipo de metodologias como **Relaxação por Bandas** [Lumsdaine88] [Dongarra87], ou bastante mais envolventes como **Solução de Ondas por Newton**, **Relaxação de Newton** ou **Métodos de Colocação** foram igualmente desenvolvidos embora sem vantagens apreciáveis. Tais algoritmos poderão no entanto futuramente assumir uma importância considerável como adiante se explicará.

Em qualquer dos métodos descritos anteriormente, a solução do circuito passa invariavelmente pela resolução de um sistema de equações, e a sua complexidade cresce sobre-linearmente com o número de nós. De facto, a solução de um circuito, digamos de dimensão n, pode ir desde uma complexidade na ordem de n^3 (eliminação gaussiana) até $n^{1.2}$ (técnicas de matrizes esparsas) [Wang81].

Um outro tipo de técnicas, procurando evitar esta explosão de esforço na resolução do problema, tem sido pesquisada recentemente. Inicialmente proposta apenas ao nível de simulação temporal, por razões de estabilidade e precisão dos algoritmos até então desenvolvidos [Kim84] [Hwang86] [Bauer87], foi mais recentemente extendida à simulação de circuitos ao nível eléctrico, através de novos algoritmos, cujas propriedades numéricas ainda estão em fase de estudo [Vidigal86] [Odryna86i] [Bauer88].

Estes algoritmos baseiam-se em técnicas de controlo por acontecimentos, até então apenas usadas em simuladores ao nível lógico, e cuja característica de traço selectivo permite de forma eficaz explorar a latência existente nos circuitos, acelerando a simulação de forma considerável embora mantendo uma precisão aceitável nos resultados obtidos. A vantagem de tal tipo de técnicas é que, nos métodos de traço selectivo, os requerimentos em termos de tempo de complexidade crescem linearmente com a actividade dos nós, o que é tipicamente sub-linear no número de nós. Tal tipo de algoritmos, pelas semelhanças que apresenta relativamente aos simuladores lógicos, parece naturalmente mais fácil de paralelizar do que os métodos anteriormente expostos, o que aumenta o seu interesse. Atendendo a que o simulador utilizado como base neste trabalho tem as características agora indicadas, as técnicas utilizadas neste tipo de algoritmos serão mais detalhadamente descritas adiante.

3.2.3 Simulação Eléctrica e Processamento Paralelo

A pesquisa no campo de simulação de circuitos tem nos últimos anos focado a sua atenção em reduzir os tempos de simulação pela exploração de paralelismo, o que se tornou possível com o aparecimento de máquinas paralelas comerciais.

Nesta secção descrevem-se de forma generalizada os esforços feitos no sentido de implementar algoritmos de simulação de circuitos em máquinas paralelas. Atenção especial é dada às implementações baseadas em metodologias tradicionais, sendo os algoritmos do tipo *event-driven* mencionados na secção seguinte. Qualquer das operações debatidas nesta secção é extremamente dependente do tipo de arquitectura a que a implementação se destina. Em simulação de circuitos, as arquitecturas tipicamente utilizadas são, ou as baseadas em *bus* com uma memória partilhada, ou estruturas do tipo distribuído, com uma memória local a cada processador e um sistema de mensagens entre eles. Mencionam-se apenas aspectos gerais aplicáveis a tal tipo de arquitecturas [Jacobs86].

Os dois componentes fundamentais que compõem a parte computacionalmente pesada da simulação eléctrica de circuitos são a composição da matriz a resolver (ou seja a linearização dos elementos do circuito) e a sua resolução. Estes dois problemas, em geral, levantam questões diferentes quanto a serem paralelizados. A paralelização explorada na construção da matriz do circuito linearizado assenta no facto de que essencialmente cada elemento pode ser linearizado independemente dos outros, requerendo apenas uma certa informação local, sendo necessário posteriormente alguma forma de sincronização no preenchimento da matriz como soma de várias contribuições. A resolução das equações do circuito é bastante mais envolvente e dependente do algoritmo utilizado.

A linearização do sistema de equações que descreve o circuito passa pela linearização das características dos elementos não lineares do mesmo. Tal é, por exemplo, o caso dos elementos semi-condutores. A linearização de tais dispositivos, requer em geral um conjunto de informação local, como sendo a tensão num grupo de nós na proximidade do dispositivo, e envolve, em geral, um conjunto extenso e complicado de operações numéricas. No entanto, essa linearização é feita elemento a elemento, o que lhe confere elevado grau de paralelismo. De facto, todos os elementos podem, simultaneamente, determinar os parâmetros resultantes da linearização das suas características num dado ponto, o que possibilita um elevado grau de aceleração em casos de circuitos com um grande número de dispositivos não-lineares. Uma vez calculados os parâmetros de interesse para todos os elementos do circuito, é necessário combiná-los de uma certa forma, para se obter a matriz linearizada do sistema. Este preenchimento dos elementos da matriz provoca um bloqueamento, uma vez que muitos dispositivos contribuirão para o mesmo elemento. É necessário, então, implementar mecanismos de sincronização que assegurem a consistência dos dados colocados em memória de forma a não se corromperem nem se perderem as várias contribuições para os elementos da matriz.

Este último ponto é nitidamente dependente da arquitectura do multiprocessador utilizado. Em sistemas baseados em *bus*, em que a memória é partilhada por todos os processadores, um sistema de sincronização utilizando semáforos pode ser implementado. Para tal associa-se com cada posição de memória um semáforo que garante a exclusividade no acesso à dita posição. Estudos realizados levam a concluir que o número de activações de tais semáforos pode tornar-se incomportável, pelo que se torna mais eficiente associar semáforos a um conjunto de posições de memória, em geral uma linha da matriz, de forma a cada dispositivo poder colocar de uma só vez a sua contribuição em diversos elementos da matriz [Sadayappan]. Em sistemas descentralizados este processo é semelhante, embora no caso geral seja de supôr que a matriz se encontra distribuida pelos vários processadores, pelo que o problema equivalente consiste em determinar um encaminhamento eficiente para as várias contribuições, de forma a minimizar a congestão nas linhas.

Em qualquer dos casos, os problemas envolvidos na paralelização do preenchimento da matriz do sistema são problemas do campo da Ciência de Computação e de Arquitectura de Computadores, respeitando a encaminhamento de mensagens, controlo de congestão, sincronização de acessos a memória, etc. Como exemplos dos vários tipos de metodologia adoptadas podem-se mencionar as várias implementações efectuadas [Gallivan] [Jacobs86] [Deutsch85] no caso de sistemas centralizados. Como exemplos de resolução do problema em arquitecturas descentralizadas podem-se mencionar o trabalho de [Webber87] [Silveira88ii].

A paralelização da resolução do sistema algébrico linear resultante da aplicação de uma das metodologias indicadas é um processo bastante mais difícil. Por um lado torna-se extremamente complexo implementar os métodos directos em arquitecturas paralelas. Isto porque a eliminação gaussiana ou a decomposição LU, não apresentam em si um grau de paralelismo suficiente para se tornarem eficazes num multiprocessador, versus a sua versão sequencial. Várias implementações de métodos directos em arquitecturas de processamento paralelo foram já efectuadas [Hoepken87] [Gallivan] [Jacob85], embora os resultados obtidos demonstrem a existência de problemas estruturais difíceis de transpôr.

Por outro lado a utilização de métodos de relaxação parece encerrar um grau de paralelismo superior, tendo sido obtidos melhores resultados com este tipo de algoritmos [Smart87].

A eliminação gaussiana ou a decomposição LU apresentam de facto características de elevada dependência entre os vários factores. A utilização de técnicas de relaxação parece conduzir na maioria dos casos a acelerações maiores, em especial quando se trata de circuitos de grande dimensão e com acoplamento fraco entre elementos ou grupos de elementos. Neste campo, foi recentemente demonstrado que algoritmos de convergência mais lenta sequencialmente, podem apresentar um grau de paralelismo superior. Tal levou ao estudo de métodos, como **Relaxação por Bandas** [Lumsdaine88], que procuram o desacoplamento das equações do circuito como forma de maximizar o paralelismo existente. Um outro ponto a considerar diz respeito ao facto de que os métodos directos não se escalam de forma apropriada com o aumento do números de processadores. O aparecimento de computadores massivamente paralelos, nos últimos anos, abriu novos caminhos na pesquisa de algoritmos paralelos, tendo-se desenvolvido metodologias que tentam explorar ao máximo o paralelismo inerente aos problemas, de forma a utilizar a enormidade de recursos postos à disposição. No caso da simulação eléctrica, desenvolveram-se recentemente novos algoritmos como **Solução de Ondas por Newton**, **Relaxação de Newton** [White85i] ou **Métodos de Colocação**, que procuram explorar de forma mais eficiente a enormidade de recursos disponíveis [White85i]. Tais métodos tentam resolver um número superior de problemas idênticos ao mesmo tempo, sendo para tal determinada a solução do circuito em vários pontos no tempo simultaneamente.

O desenvolvimento de algoritmos de simulação em arquitecturas paralelas está assim numa fase embrionária, sendo no entanto ideia generalizada de que só com algoritmos inovadores se conseguirão factores de aceleração que tornem a simulação paralela verdadeiramente interessante.

3.3 Metodologia de Simulação em CINNAMON

O simulador **CINNAMON** [Vidigal86], actualmente em desenvolvimento pelo grupo de simulação do **INESC**, baseia-se em técnicas recentes de controlo de simulação, no sentido de atingir um elevado desempenho. O objectivo fundamental do simulador é o de possibilitar a simulação de circuitos de grandes dimensões, com elevada precisão e em tempo aceitável. Para tal tira-se partido da latência existente nos circuitos, de forma a aumentar dramaticamente a velocidade de simulação.

Os pontos essenciais do algoritmo, e as técnicas em que se baseia, foram detalhadamente descritos nos artigos originais [Vidigal86] [Neto87i] pelo que aqui apenas se mencionarão brevemente as ideias básicas do algoritmo.

3.3.1 O Algoritmo de CINNAMON

CINNAMON é um simulador em que a progressão da simulação se baseia na resolução de acontecimentos que correspondem a perturbações nos nós do circuito. Ao contrário da maioria dos simuladores convencionais, mesmo os que empregam um sistema de traço selectivo, **CINNAMON** baseia-se em técnicas de discretização do eixo das tensões e não do eixo dos tempos. Um acontecimento corresponde a uma perturbação que provoca uma alteração num nó, causando no mesmo uma possível variação de tensão. Sempre que a tensão num nó varia entre dois níveis de tensão permitidos, é necessário resolver o acontecimento associado a essa variação, ou seja, predizer as consequências futuras dessa variação no comportamento do circuito.

CINNAMON utiliza uma formulação em termos do método nodal, construído em torno das tensões nos nós do circuito, o que como se viu conduz ao sistema de equações diferenciais:

$$C(v(t), i(t))\,\frac{d}{dt}v(t) + G(v(t), i(t))\,v(t) = i(t) \tag{3.11}$$

em que $v(t)$ é o vector das tensões nos nós do circuito, $i(t)$ o vector das fontes do circuito, e $C()$ e $G()$ respectivamente as matrizes dos elementos capacitivos e resistivos presentes em cada nó.

A formulação anterior implica que, ao contrário da maioria dos métodos apresentados anteriormente, o sistema que se constrói é um sistema diferencial linear, uma vez que as matrizes $C()$ e $G()$ são a soma das contribuições dos elementos lineares com os elementos não-lineares, após estes terem sido linearizados em torno da solução actual. Este tipo de metodologia, se bem que menos precisa que a apresentada anteriormente, tem à partida o conveniente de não conduzir a sistemas não lineares e logo evitar a utilização de algoritmos do tipo Newton-Raphson, que são normalmente bastante dispendiosos. No entanto, a aplicação de um método de integração apropriado, conduz ainda, a um sistema de equações cuja dimensão é da ordem do número de nós do circuito[3]. A resolução desse sistema, como debatido anteriormente, é um processo computacionalmente muito consumidor.

[3]Mais exactamente, igual ao número de nós do circuito menos um (a referência).

Torna-se necessário neste ponto determinar um método de integração apropriado para a resolução do sistema de equações diferenciais anterior, bem como um algoritmo para a resolução do sistema algébrico daí resultante. Para motivar a solução seguida em **CINNAMON** para os problemas anteriores, considere-se a equação para o nó k, extraída do sistema (3.11):

$$c_{kk}\ \frac{d}{dt}v_k(t) + g_{kk}\ v_k(t) - \sum_{j\neq k} c_{kj}\ \frac{d}{dt}v_j(t) - \sum_{j\neq k}\ g_{kj}\ v_j(t) = i_k(t) \tag{3.12}$$

em que os somatórios são efectuados para todos os nós do circuito exceptuando k, c_{kj} e g_{kj} referem-se respectivamente à capacidade e conductância que liga os nós k e j, $i_k(t)$ é a soma das correntes provenientes de fontes, que entram no nó k e c_{kk} e g_{kk} são, respectivamente, a soma das capacidades e conductâncias ligadas ao nó k.

Um dos conceitos mais interessantes desenvolvidos em **CINNAMON** é o conceito de **actividade localizada** introduzido de seguida.

Definição 3.1 (Actividade Localizada)

Uma perturbação num dado instante de tempo, num dado ponto de um circuito demora um tempo não nulo a propagar-se a outras zonas do circuito.

Este conceito, que se relaciona directamente com a latência existente nos circuitos, apela à introdução da definição de **esfera de influência de um nó**.

Definição 3.2 (Esfera de Influência de um nó)

Esfera de Influência de um nó é o conjunto mínimo de nós e dispositivos, que de alguma forma são afectados directa e imediatamente por uma alteração na tensão desse nó.

Considere-se de novo a equação 3.12 e suponha-se que no instante t o nó k sofreu uma alteração no valor da sua tensão. Esta equação pode ser encarada como uma fórmula explícita para determinar $v_k(t)$, se se utilizarem como aproximações para o valor dos outro nós, v_j, e suas derivadas, $\frac{d}{dt}v_j$, a informação mais actual existente. Numa outra perpectiva isso corresponde a dizer que os termos $\sum_{j\neq k} c_{kj}\ \frac{d}{dt}v_j$ e $\sum_{j\neq k} g_{kj}\ v_j$ são negligenciáveis face a i_k, ou, de forma mais precisa, a considerar que as tensões dos nós na esfera de influência do nó k não variam de forma significativa no intervalo de tempo sob consideração.

Se bem que aparentemente sensata, a aproximação anterior carece de alguma validação. De facto, por um lado, nas topologias dos circuitos MOS actualmente fabricados, é típico verificar-se que a conectividade de um nó (número de nós ligados a um nó) é poucas vezes superior a 2 e raramente superior a 4. Por outro lado, e baseado no conceito de actividade localizada anteriormente introduzido, a esfera de influência de um nó pode, com razoável precisão, ser aproximada ao conjunto de nós directamente ligados a esse nó, dado que num intervalo de tempo pequeno, após uma perturbação num dado nó, apenas os nós que lhes estão directamente ligados tiveram oportunidade de sentir essa perturbação.

Prosseguindo então no sentido indicado pela aproximação introduzida, somos conduzidos a um sistema de equações desacopladas, próprio de um método explícito como o enunciado, que tem a forma geral:

$$c_{kk}\ \frac{d}{dt}\hat{v}_k(t) + g_{kk}\ \hat{v}_k(t) = \hat{\imath}_k(t) \tag{3.13}$$

que não é mais do que uma equação diferencial de primeiro grau, cuja solução exacta[4] é:

$$\hat{v}_k(t) = (\hat{v}_{ko} - \hat{v}_{k\infty})e^{(t-t_0)/\tau_k} + \hat{v}_{k\infty} \tag{3.14}$$

em que $\hat{v}_{k\infty} = \hat{\imath}_k/g_{kk}$, $\tau_k = c_{kk}/g_{kk}$ e $\hat{v}_{ko} = \hat{v}_k(t_0)$

Sempre que a aproximação introduzida for considerada válida, é então possível determinar o instante de tempo em que o nó k atingirá o próximo valor discreto de tensão $(\hat{v}_{ko} \pm \Delta V)$. Isso sucederá de facto no instante de tempo:

$$t_{next} = t_0 - \tau_k \ln(\frac{\hat{v}_{ko} - \hat{v}_{k\infty} \pm \Delta V}{\hat{v}_{ko} - \hat{v}_{k\infty}}) \tag{3.15}$$

no qual, por definição, se terá um novo acontecimento no nó k. Note-se que de acordo com o atrás exposto, a resolução de um acontecimento no nó k implica alterações na esfera de influência desse nó, pelo que a alteração relacionada com o acontecimento tem de ser propagada para os nós adjacentes do nó k, levando à marcação de novos acontecimentos para esses nós.

É agora possível tornar mais claro o conceito de **acontecimento**.

[4]Note-se que o que aqui se chama solução exacta da equação diferencial não é a solução exacta da função, $v_k(t)$ atendendo às aproximações anteriormente introduzidas.

Definição 3.3 (Acontecimento)

Um acontecimento corresponde a uma alteração de $\pm\Delta V$ no valor da tensão de um nó do circuito. Essa alteração está naturalmente ligada a uma perturbação que se propaga ao longo do circuito.

Desta forma a simulação prossegue baseada na propagação de perturbações associadas a acontecimentos nos nós do circuito, até que este atinge um estado estável em que não haverá mais acontecimentos marcados.

3.3.2 Análise do Simulador CINNAMON

A parte numérica do algoritmo utilizado no simulador **CINNAMON** foi já descrita. Torna-se necessário agora abordar um pouco das características específicas da implementação sequencial na qual este trabalho se baseou.

Sendo baseado numa metodologia de controlo por acontecimentos, a estrutura fundamental utilizada no processo de simulação é uma lista ligada, correspondente às marcações existentes para os vários nós. Um nó está na lista quando está marcado um acontecimento para esse nó, num dado instante de tempo. Cada nó apenas pode estar uma vez na lista de acontecimentos, dado que não são permitidas variações superiores ao passo de discretização das tensões. A informação contida em cada elemento desta lista é a necessária para processar um acontecimento, ou seja, o nó a que o acontecimento se reporta, o instante de tempo em que este vai ser processado e o nível de tensão que o nó em causa vai atingir. A resolução de um acontecimento corresponde assim a, nesse instante de tempo, retirar o elemento da lista, actualizar a tensão no nó e propagar a variação processada aos nós vizinhos.

A simulação é assim controlada pelo fluxo da lista de acontecimentos, sendo considerada terminada quando esta lista fica vazia, ou seja, não há mais acontecimentos marcados.

Em termos gerais, o algoritmo de controlo da simulação, para a versão sequencial, é o seguinte[5]:

[5] O apêndice B consiste numa versão um pouco mais detalhada do algoritmo.

Algoritmo 3.2 (Cinnamon: Versão Sequencial)

```
simulação()            /* seja DC ou Transiente */
{
    t = 0 ;
    marcação das fontes de tensão;
    enquanto( há acontecimentos na lista e t < TempoMáximo) {
        lê a informação do próximo acontecimento (nó, tempo, tensão) ;
        t = tempo ;
        lineariza os dispositivos ligados ao nó activo;
        determina a próxima alteração (nó, lista de nós adjacentes) ;
        guarda informação do acontecimento (nó, tempo, tensão) ;
    }
}
```

O algoritmo descrito representa a estrutura actual do simulador. Os resultados obtidos até ao momento são altamente encorajadores, tendo-se registado tempos de simulação cerca de 2 a 3 ordens de grandeza inferiores aos de simuladores convencionais como SPICE [Nagel75], mantendo uma precisão semelhante.

É importante notar de momento dois pontos importantes: por um lado a discretização do eixo das tensões imposta pelo algoritmo na versão actual limita bastante a utilidade do método para circuitos analógicos puros. Por outro lado, a eficiência das técnicas utilizadas varia com o circuito que se está a simular, particularmente com o seu tamanho e actividade. Na verdade, é claro hoje em dia, que para circuitos de pequenas dimensões ou de reduzida actividade é difícil bater os simuladores utilizando métodos directos, devido sobretudo ao elevado estado de refinamento das técnicas que estes utilizam. No entanto, verifica-se experimentalmente, que mesmo nestes casos menos favoráveis, **CINNAMON** consegue acompanhar os simuladores convencionais em termos de desempenho e precisão.

3.3.3 Desenvolvimento de CINNAMON

O projecto de desenvolvimento de **CINNAMON**, no grupo de Simulação de Circuitos do Projecto de MicroElectrónica do INESC, iniciou-se em meados de 1986. O código inicial foi substancialmente reescrito desde então, essencialmente por razões de desempenho. Foi alterada a estrutura de dados, de forma a torná-la mais flexível, tendo sido adicionada funcionalidade ao simulador.

Neste momento o desenvolvimento do simulador passa por um conjunto de vertentes um pouco diferenciadas mas igualmente importantes. Dentre elas podem-se distinguir as seguintes:

- Estudo das características numéricas do algoritmo
- Desenvolvimento de uma versão com passo de tensão variável
- Estudo de Macromodelos para estruturas funcionais
- Estudo de alternativas à aceleração do algoritmo
- Adição de modelos de dispositivos mais elaborados

O estudo numérico do algoritmo está a decorrer nomeadamente no sentido de provar a convergência matemática do algoritmo. Ao contrário de outras tentativas semelhantes, feitas anteriormente [Kim84] [Hwang86], e prosseguidas recentemente [Bauer88], cujas propriedades de estabilidade como algoritmo de integração deixam algo a desejar, o algoritmo numérico de **CINNAMON** é estável e, aparentemente, converge para o resultado correcto, desde que a discretização de tensões seja apropriada. Tal no entanto não foi ainda demonstrado, embora os resultados experimentais pareçam confirmá-lo.

Talvez uma das partes mais importantes e interessantes do desenvolvimento actual do simulador é a pesquisa que tem sido conduzida no sentido de possibilitar a utilização de uma discretização de tensões variável e adaptativa [Neto87ii]. Os resultados das tentativas até agora efectuadas indicam que tal não só é possível, como possibilita igualmente a introdução de um método de previsão do erro cometido pelo algoritmo na resolução de cada acontecimento, pelo facto de se estarem a desacoplar as equações [Neto89]. A possibilidade de utilizar valores de discretização das tensões diferentes para cada nó em cada instante e, sobretudo, a forma adaptativa e automática como tal se deverá processar, aumentarão grandemente a utilidade do simulador, dado que passará a ser possível simular circuitos puramente analógicos com grande eficiência.

Uma outra vertente do desenvolvimento actual de **CINNAMON** diz respeito à adição de modelos de dispositivos mais elaborados. O código relacionado com modelização foi reescrito, pelo que se torna actualmente extremamente simples adicionar novos modelos com um mínimo de esforço [Guimarães87]. Adicionalmente, têm-se estudado técnicas de cálculo dos parâmetros dos transistores baseadas em tabelas previamente compiladas. Este processo, que permitirá decerto um aumento da eficiência do simulador, sobretudo se atendermos à complexidade dos modelos dos dispositivos semi-condutores, torna-se ainda mais interessante em termos futuros. De facto, o constante progresso tecnológico dos processos de fabricação possibilita, hoje em dia, a utilização de processos extremamente difíceis de modelar e cuja forma mais correcta de simular é utilizando conjuntos de dados obtidos directamente por experimentação. Este campo de pesquisa deverá tornar-se ainda mais importante nos anos mais próximos.

A simulação multi-nível foi já mencionada na introdução desta tese. Trata-se, se efectuado de forma automática, de um processo que decerto permitirá a simulação de sistemas completos em tempo reduzido, com a precisão necessária à verificação de cada bloco do circuito. A simulação multi-nível ganha interesse crescente na medida em que as técnicas de desenho de circuitos, em uso actualmente, são sempre hierárquicas, dado o volume de informação envolvido. Assim, em cada projecto novo, incorporar-se-ão blocos já utilizados e detalhadamente testados em projectos anteriores, os quais não é necessário testar com o mesmo detalhe do resto do circuito, mas têm mesmo assim de ser incluídos na simulação do novo sistema. A solução reside em definir um interface correcto para estes blocos, baseado na sua funcionalidade e nas suas características, de forma a se obter um interface simples com os blocos recém-desenhados [Teixeira86] [Teixeira87].

Finalmente, convirá notar que a pesquisa no campo da aceleração de algoritmos de simulação eléctrica de circuitos, em particular relativos ao simulador **CINNAMON**, continua, estando-se neste momento a considerar outro tipo de arquitecturas, de grão mais grosso, para as quais se torna necessário o desenvolvimento de algoritmos de partição de um circuito em blocos. Este tema será debatido adiante.

3.3.4 Sequencialidade e Paralelismo em CINNAMON

As técnicas de simulação utilizadas em **CINNAMON** e descritas anteriormente são extremamente eficientes na medida em que, como vimos, exploram a latência existente num circuito de forma dinâmica. A forma natural como tal é feito, por discretização do eixo das tensões, pode ser entendida como um processo automático de ajustamento do passo de integração da equação diferencial.

O conceito de **actividade localizada**, anteriormente introduzido, conduz como vimos, a um processo extremamente simplificado de resolução do sistema de equações diferenciais não lineares que descreve o circuito. De facto, o processo de controlo da simulação por acontecimentos, derivado da aplicação do conceito de actividade localizada, leva-nos em cada instante a considerar apenas um sub-circuito como estando activo, e logo a resolver apenas um sub-conjunto do sistema de equações completo. A metodologia seguida para resolução de um acontecimento leva aliás a um desacoplamento completo das equações em cada instante de tempo, evitando assim os problemas algébricos encontrados pela maioria dos algoritmos.

Torna-se claro no entanto, que dentro das condições mencionadas, havendo actividade simultânea em várias zonas de um circuito, em nós cujas esferas de influências não se intersectam, tais acontecimentos podem ser resolvidos sem interferência mútua. É portanto possível acelerar o algoritmo de simulação, na condição que acontecimentos não correlacionados[6] possam ser resolvidos simultaneamente. Tal implica necessáriamente a existência de várias unidades de processamento, e logo uma arquitectura capaz de, de alguma forma, aproveitar o paralelismo inerente ao algoritmo. Motiva-se desta forma o estudo de algoritmos de paralelização da metodologia do simulador.

O algoritmo de **CINNAMON**, na forma descrita, tem de facto duas características que o tornam adequado a implementação em arquitecturas paralelas. Uma é o processo de controlo da simulação por acontecimentos: a existência de um processo despachante *(scheduler)* que controla a evolução da simulação, gerindo a lista de acontecimentos, origina um processo dinâmico de detecção e exploração da latência do circuito. A outra deriva do facto de que, para cada acontecimento, apenas um dado nó e aqueles que lhes estão directamente ligados, necessitam de ser reavaliados, o que limita a resolução de um acontecimento apenas à solução de um sistema diagonal de pequena dimensão.

Note-se que o processo agora descrito não é tão transparente como possa parecer de início. De facto, se dois acontecimentos, em sequência na lista, se referirem a nós não directamente ligados, é possível resolvê-los em simultâneo, mas a resolução de um deles pode implicar que o acontecimento marcado para o outro seja desmarcado ou remarcado. De facto, considere-se a lista de acontecimentos como apresentada na figura 3.1 e suponha-se que existe um método de determinar se o acontecimento B pode ser resolvido em simultâneo com A. Supondo que tal é possível, isso indica que B não está na esfera de influência de A, podendo no entanto estar fortemente ligado a A. Nesse caso, é possível que a resolução do acontecimento A venha a originar uma marcação de um nó vizinho de A e de B, que pode provocar alteração nas marcações já existentes para B. Da mesma forma, a simples resolução de A pode originar novos acontecimentos temporalmente anteriores a B, pelo que não há garantia que B se venha sequer a realizar.

Os dois efeitos mencionados anteriormente são de facto extremamente indesejáveis, mas têm a virtude de pôr a claro um ponto importante para uma futura implementação. Apesar de utilizar técnicas de controlo por acontecimentos similares às mencionadas para

[6] A palavra correlação é utilizada aqui de uma forma imprecisa e lata não devendo ser considerada no sentido estatístico usual.

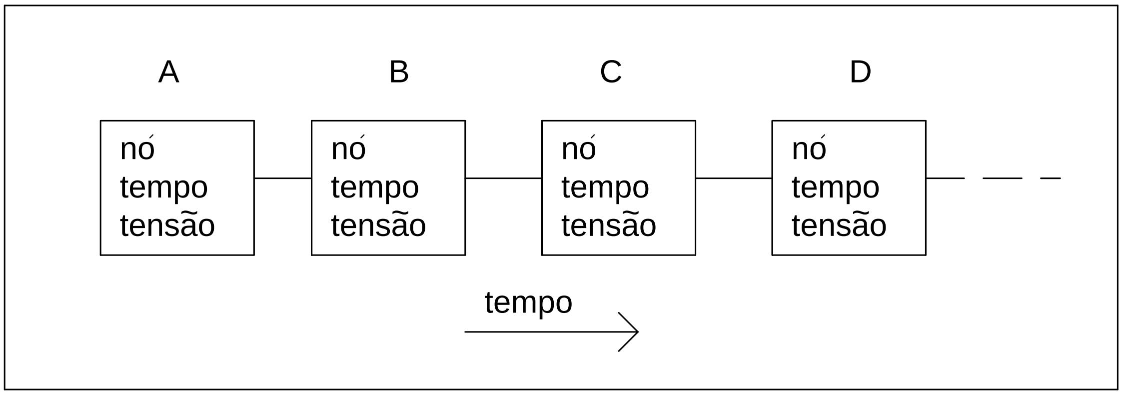

Figura 3.1: Lista de acontecimentos para a simulação de um dado problema

os simuladores de nível lógico, **CINNAMON** é claramente distinto daqueles, no sentido em que opera com tempo contínuo[7] e não em unidades de um atraso mínimo. Desta forma, não é possível no caso geral aproveitar a simultaneidade temporal de vários acontecimentos para explorar paralelismo, como é o caso dos simuladores lógicos. Por outro lado é importante apontar uma outra diferença notável entre os dois tipos de simuladores. Num simulador lógico, assume-se unidireccionalidade no modelo de um elemento (porta, elemento de memória, etc), o que não é de todo verdade neste caso. Assim, e relembrando que a resolução de um acontecimento implica recalcular um conjunto de nós numa dada vizinhança, os acontecimentos atribuídos a esses nós podem trazer perturbações ao nó inicial. Em geral, esse será o caso.

Foram desta forma apresentados de forma simples, os dois problemas mais salientes de uma implementação paralela de **CINNAMON**. Em primeiro lugar é necessário um processo de determinar se dois acontecimentos, em sequência na lista, podem ser resolvidos em simultâneo, o que não pode ser nem demasiado restritivo nem permissivo. Em segundo lugar é necessário, porventura, recuperar o estado das variáveis do circuito caso um acontecimento já resolvido venha a ser remarcado para uma outra altura, ou mesmo desmarcado.

A solução do primeiro dos problemas mencionados indica a necessidade de um algoritmo de partição do circuito em blocos suficientemente desligados, para poderem ser resolvidos em paralelo quaisquer acontecimentos relativos a nós interiores a essas partições. A resolução do segundo problema implica a existência de um mecanismo de arquivo do processo de simulação, cujo conteúdo possibilite recuperar estados anteriores.

[7]Tal não é de facto completamente verdadeiro devido à precisão finita da representação numérica num computador, mas a essência do problema mantêm-se dado que essa precisão é elevada.

A implementação do simulador num multiprocessador parece requerer desta forma a existência de um algoritmo que particione o circuito. Esta partição pode ser efectuada estática ou dinamicamente. Partições estáticas são realizadas por um pré-processador que lê a descrição do circuito e fornece um conjunto de sub-circuitos obedecendo a determinados critérios de separação. Este tipo de partições, se bem que simples de implementar, pode a médio prazo tornar-se pouco eficiente. Isto sucede porque qualquer tipo de partição estática baseia-se apenas na topologia do circuito, não tendo qualquer informação sobre a dinâmica e actividade do mesmo, o que pode tornar o acoplamento entre sub-circuitos muito diferente do que a topologia indica.

Partições dinâmicas, pelo seu lado, são consideravelmente mais difíceis de implementar e consomem uma percentagem maior do tempo de computação, mas dada a informação a que se podem reportar ser sempre mais actualizada, podem-se tornar mais eficientes a longo prazo. A questão de eficiência aqui pode-se pôr em termos do equilíbrio da carga computacional dos vários processadores. De facto, a obtenção de aceleração de um algoritmo numa máquina paralela, implica claramente a minimização do tempo em repouso de cada processador. Neste ponto portanto, as partições dinâmicas, se implementadas de forma eficiente podem proporcionar um melhor desempenho.

A existência de um equilíbrio de esforço entre os vários processadores é de importância capital para a obtenção de um bom desempenho. A utilização dos recursos disponíveis tem de ser maximizada, e para tal é necessário um esquema de divisão de tarefas pelos processsadores, baseado nesse objectivo. A existência de sub-circuitos derivados do circuito original e a sua assignação aos diversos processadores tem de ser estudada cuidadosamente. Este processo é no entanto dependente da arquitectura da máquina que se utilize (excepto nos casos em que o número de partições é igual ao número de processadores). O controlo da interação entre as várias tarefas é igualmente um factor a ponderar, podendo escolher-se entre um esquema síncrono do tipo do utilizado em máquinas SIMD, ou um esquema assíncrono mais geral.

Em qualquer dos casos, a arquitectura do multiprocessador utilizado condiciona de forma decisiva a maioria das opções a ter em conta. Endereçaremos aqui o problema de utilização das arquitecturas típicas comercialmente existentes: as máquinas de memória partilhada baseadas em *bus* e as máquinas descentralizadas com estrutura de hipercubo ou outra.

Nas máquinas de memória partilhada, a manutenção da consistência dos dados da simulação é delicada, dado que acontecimentos em tempos diferentes estarão a ser simulados em simultâneo e o tempo de resolução de cada acontecimento não será o mesmo (dado ser dependente da conectividade de cada nó).

Para este tipo de máquinas, a partição pode ser realizada indiferentemente de forma estática ou dinâmica, dado que todos os dados estão na memória comum, podendo no entanto as partições dinâmicas conduzir a um melhor desempenho. Em qualquer dos casos, a melhor opção parece ser a de atribuír cada subcircuito ao processador que primeiro ficar livre, o que equivale a uma assignação do tipo "aleatório"[8] como sendo a mais correcta.

A segunda classe de máquinas comercialmente disponíveis são máquinas descentralizadas, utilizando um esquema de transferência de dados indirecto. Neste caso, não é mais possível pensar em termos de uma lista de acontecimentos única, pois isso implica um controlador global, o que nestas máquinas se pretende evitar. Se no entanto o circuito for dividido de forma a que o equilíbrio entre a carga dos diversos processadores seja mantido e, ao mesmo tempo, a alocação de sub-circuitos com vizinhanças comuns seja feita a processadores directamente ligados, este problema pode ser resolvido. No entanto este ponto é extremamente delicado. Por um lado pode ser extremamente difícil satisfazer as duas imposições anteriores, dado que a distribuição espacial dos dados pelos processadores, não conduz necessariamente a um bom balanceamento da carga computacional. Neste caso, torna-se necessário encaminhar mensagens através de vários processadores, o que implica uma menor eficiência.

A estratégia de partição a implementar neste tipo de máquinas deve minimizar o número de sub-circuitos directamente ligados, de forma a diminuir as necessidades de comunicação. Claramente a estratégia de alocação daqueles aos vários processadores não será aleatória, mas antes tentará mapear as ligações físicas do circuito às ligações físicas da máquina. Uma alocação por traço, como discutido anteriormente, pode constituir uma boa alternativa.

Necessariamente, o tipo de partição a implementar terá de ser inicialmente estático, podendo no entanto, para equilibrar a carga dos vários processadores, assistir-se a reagrupamentos ou repartições do circuito, o que no entanto implica migração de dados entre processadores, logo diminuindo a eficiência global.

Neste tipo de máquinas, e devido à sua natureza descentralizada, um algoritmo rigidamente síncrono não será muito eficiente, a menos que existam mecanismos para assegurar esse sincronismo aos nível da execução dos processadores (como em máquinas do tipo SIMD). Neste caso parece claro que se a partição tiver sido correcta, o controlo síncrono da simulação é a forma mais correcta de resolver o problema. Caso contrário, uma vez que um controlo síncrono descentralizado é muito intensivo do ponto de vista de comunicação, é preferível utilizar estratégias assíncronas. Isto implica, no entanto, a existência,

[8] O *scheduling* de sub-circuitos a processadores pode igualmente ser visto como uma estrutura do tipo FIFO: primeiro a entrar-primeiro a sair.

em cada processador, de mecanismos que permitam retroceder a simulação no tempo, pois se os processadores executarem assincronamente, como a lista de acontecimentos é distribuida, pode suceder que certos acontecimentos já resolvidos tenham de ser alterados. Este processo pode tornar-se extremamente complicado uma vez que os necessários mecanismos de recuperação de um estado anterior, podem tornar-se extremamente difíceis de manter, devido à distribuição dos dados. Acresce que o processamento implica a existência de relógios que disparem ao fim de um dado tempo, se certos dados não se tornarem disponíveis, de forma a evitar situações de *deadlock* (bloqueamento indesejável da sequência do programa).

Neste tipo de arquitecturas, o problema da comunicação, entre processadores que desejam trocar informação, é potencialmente a maior fonte de perda de eficiência, pelo que os algoritmos de partição de um circuito devem, neste caso, ser atentamente ponderados.

Em tudo o que se disse anteriormente assumiu-se como premissa que o paralelismo inerente ao algoritmo de **CINNAMON** é um paralelismo espacial ao nível dos dados. Além disso descreveram-se apenas as opções mais importantes de um algoritmo de grão grosso (*coarse-grain*). De facto, o próprio algoritmo do simulador, ao desacoplar completamente as equações do circuito em cada instante de tempo, sugere que um algoritmo de grão fino (*fine-grain*) pode ser implementado, sobre uma partição fina em que cada sub-circuito consiste apenas num nó do circuito. Acresce que o esquema de controlo por acontecimentos, oferece uma forma automática de determinação dos sub-circuitos activos. Este tipo de algoritmo é extremamente interessante na medida em que não tem associado qualquer tempo de inicialização[9]. As implementações do tipo grão fino parecem apropriadas para arquitecturas do tipo descentralizado, em particular com um número de processadores elevado. Em arquitecturas baseadas na existência de uma memória comum, experimentarão, no entanto, problemas de contenção no *bus* devido às grandes necessidades de comunicação que lhes estão inerentes.

Além do paralelismo espacial mencionado, é igualmente possível explorar paralelismo temporal no algoritmo do simulador. Este é de certa forma oferecido pela característica de controlo por acontecimentos, mas pode ser refinado, na medida em que a resolução de cada um desses acontecimentos pode ser subdividida num conjunto de operações numa dada sequência. Um exemplo seria subdividir o ciclo de simulação numa *pipeline* de funções como por exemplo:

- escrita das equações.

[9] Uma vez que cada partição é constituída por um nó não é, na prática, necessário efectuar qualquer partição.

- linearização de elementos.
- reavaliação do nó que causou o acontecimento.
- reavaliação dos nós na sua esfera de influência.

Este processo, totalmente síncrono, não é expansível para um número elevado de processadores, a menos que se explore simultaneamente paralelismo espacial. Pode no entanto tornar-se interessante em máquinas com poucos processadores e onde seja simples sincronizar tarefas. Tal aproximação seria baseada numa partição funcional de grão grosso. Note-se que neste caso, podemos ter um acontecimento na fase final de resolução e outro já em fase de inicialização. Se neste caso o primeiro acontecimento originar uma remarcação do segundo, isso implica esvaziar a *pipeline* e recomeçar de novo. Desta forma também aqui é necessário manter arquivos dos estados do sistema para poder retroceder no tempo até um estado anterior a um acontecimento já na *pipeline*.

O problema mais importante de entre os já abordados é, ao nível da simulação, a manutenção da consistência dos dados. A sua contrapartida ao nível arquitectónico são as necessidades de comunicação dos algoritmos. Como se viu a simulação de acontecimentos em paralelo (seja funcionalmente ou não) pode originar que parte do esforço realizado tenha sido inútil e obrigue a repôr o estado do circuito num dado instante. Tudo isto deve ser efectuado numa largura de banda mínima, uma vez que se pretende minimizar a comunicação, sempre lenta, entre processadores[10].

A estratégia utilizada para manter um registo da simulação (*checkpoint*) e o mecanismo de recuperação do estado da simulação num instante no passado (*rollback*), tornam-se assim críticos neste tipo de implementações, uma vez que têm de ajustar critérios à partida contraditórios: a necessidade de guardar informação sobre o processo de simulação quando este decorre, e a perda de eficiência decorrente dessa tarefa marginal.

Qualquer implementação de um simulador, e em particular de **CINNAMON**, tem de ser ponderada à luz das opções descritas.

[10] A velocidade da transferência de dados entre processadores é geralmente várias ordens de grandeza superior a uma operação aritmética. Em máquinas de memória partilhada, tal pode não ser necessariamente verdade, mas a contenção no acesso à mesma torna esse acesso mais lento.

3.4 Arquitectura do Multiprocessador Utilizado

O Processador Paralelo que serviu de base aos protótipos descritos nesta tese é um **Alliant FX/4** instalado no **INESC** correndo um sistema de exploração da família UNIX.

A escolha de uma tal máquina para servir de base ao desenvolvimento dos conceitos que se descrevem neste trabalho, decorreu de dois factores distintos. Por um lado o preço actual dos computadores paralelos é ainda suficientemente elevado para ser impossível a aquisição de várias máquinas. Acresce que, como se verá adiante, o *Alliant* possui características que o tornam interessante como plataforma de desenvolvimento de algoritmos paralelos, mesmo que destinados a máquinas com outro tipo de arquitecturas. Por outro lado, e no contexto global do projecto ESPRIT, existem outras tarefas, a cargo do grupo do Centro de Microelectrónica Belga, que incluem o estudo de algoritmos paralelos. Esse estudo, iniciado sobre uma máquina construida pela própria Philips [Philips], será continuado sobre um **Balance 800**, um computador arquitectonicamente semelhante ao Alliant, o que facilita o trabalho de integração das ferramentas.

3.4.1 Características Arquitectónicas

O *Alliant* instalado no INESC é um mini-supercomputador com uma arquitectura paralela e com potencialidades vectoriais. Em termos arquitecturais, trata-se de uma máquina da categoria **MIMD** na taxonomia apresentada, utilizando um esquema de memória partilhada e sendo baseada num *bus* que liga os vários processadores, e estes com a memória. Trata-se igualmente de um computador com possibilidades vectoriais, concebidas para a aceleração de algoritmos numéricos beseados em estruturas de dados rígidas [Hwang84] [Silveira87i]. Com um total de 6 processadores, o *Alliant* é uma máquina vectorial e paralela de elevado desempenho [AlliantFX].

A configuração da máquina inclui um total de 4 **Elementos Computacionais** (expansível a um máximo de 8) ou **CE**, ligados através de uma rede de interconexão baseada em interruptores de encaminhamento de alta velocidade, a memórias cache de acesso rápido, para assegurar acesso rápido aos dados e código. Desta forma, é mantida a coerência da memória visível por cada processador, sendo possível obter até um pico máximo de 5 milhões de operações por segundo, envolvendo números reais, na resolução do teste Linpack[11] (dados do fabricante).

[11] Linpack é um pacote numérico destinado à resolução de sistemas de equações algébricas lineares, de distribuição gratuita, e que é geralmente utilizado como teste de desempenho de uma máquina.

Dois processadores adicionais (expansível a doze), os **Processadores Interactivos** ou **IP**, ligados através de caches à memória principal, tratam de todo o processamento interactivo, mantendo assim um elevado nível de disponibilidade e resposta, deixando aos **CE** os trabalhos computacionalmente intensivos. Estes processadores têm como função essencial o executar processos interactivos com os utilizadores do sistema, tarefas do sistema operativo criados por processos executando nos **CE**, como entradas/saídas, e actividades gerais de sistema, como paginação, etc.

Um número arbitrário de **CE** pode ser definido, quando a máquina é activada, para formar um complexo computacional em que tarefas paralelas vão ser executadas. Neste modo de funcionamento, todos os **CE** definidos como sendo parte do complexo, podem correr tarefas paralelas. Existe no entanto um outro modo, em que o complexo é formado, mantido durante um intervalo de tempo determinado e depois disperso, de forma a que todos os **CE** fiquem independentes durante um certo intervalo de tempo. Este procedimento é repetido periodicamente. A definição do tempo de activação e desactivação do complexo pode ser definido e modificado em tempo real, sem necessidade de passar pela fase de inicialização da máquina. Igualamente, em ambos os modos, nem todos os **CE** necessitam de fazer parte do complexo; alguns podem continuar independentes, sendo essa decisão deixada a cargo do supervisor do sistema de acordo com as necessidades computacionais locais. A configuração da máquina existente no INESC estabelecia um modo dinâmico, com três **CE** no complexo.

Se bem que o desempenho mencionado pelo fabricante possa ser considerado excepcional, a máquina foi essencialmente desenvolvida para cálculo científico numericamente intensivo, mais do que como um computador de uso geral, multi-utilizador e multi-processador. Tal diferença é clara a partir do número de possibilidades deixadas ao programador para aproveitar o paralelismo existente.

3.4.2 Linguagens e Potencialidades Disponíveis

Tratando-se de uma máquina essencialmente destinada a um mercado de computação científica e numérica extremamente generalizado, o *Alliant* dispõe de um compilador de "FORTRAN" altamente eficiente. Este permite o uso das potencialidades vectoriais e de paralelismo existentes, para acelerar código de outra forma consumidor de muito tempo de execução. As facilidades providenciadas a este nível não estão no entanto directamente acessíveis ao programador e implicam um paralelismo de grão extremamente fino (ao nível da instrução, por exemplo optimização de ciclos DO, etc). Um compilador de "C" com potencialidades semelhantes esteve anunciado, tendo apenas recentemente sido introduzido, pelo que não foi utilizado neste trabalho.

Um programador de "C" tem acesso a uma função que permite a execução de rotinas em paralelo. Um conjunto de funções vectoriais e concorrentes é também oferecido para cálculo científico [AlliantC]. Mecanismos ao nível do sistema operativo suportando a criação de zonas de memória partilhada, métodos usuais de sincronização e semáforos de operação indivisível (ou atómica) estão também disponíveis como ferramentas de desenvolvimento de uma aplicação paralela.

Convirá neste ponto esclarecer que os protótipos a que esta tese se reporta e que, como foi dito, consistem em implementações do simulador **CINNAMON**, foram escritos em "C", na medida em que o código original do simulador estava igualmente escrito nessa linguagem.

Uma única função do sistema permite ao programador tirar partido da concorrência existente no sistema. É a função **concurrent_call()** que toma a forma:

$$concurrent\_call(call\_flag \mid quit\_flag, func, n, arg_1, \ldots, arg_n)$$

em que "func" é a função a ser executada concorrentemente, n é o número de activações que se pretendem dessa função e o conjunto $arg_1, \ldots, arg_n$ são esses argumentos.

Um processo executando um programa escrito em linguagem "C" que chama uma **concurrent call**, cria "n" tarefas, dentro do contexto desse processo, e de tal forma que cada tarefa é assignada a um processor. A função "func" é típicamente como se descreve na figura 3.2.

Note-se que "func" recebe como primeiro argumento extra, o número da chamada concorrente que está a processar. Isto corresponde a uma numeração das tarefas desde 0 a $n-1$, o que corresponde à convenção tradicional em "C".

Existem três processos básicos de activar concorrência chamando uma **concurrent call** (as constantes seguintes estão definidas em ficheiros do sistema, nomeadamente em *"/usr/include/cncall.h"*[12]). Esses mecanismos dizem respeito ao controlo da activação de tarefas e são:

- se *"call_flag"* = *"CNCALL_COUNT"* : o sistema cria "n" tarefas para executarem no complexo, que são iniciadas assim que um dos processadores do mesmo ficar disponível e as tarefas de número inferior estiverem já terminadas ou em fase de execução.

[12] Para o leitor menos conhecedor, tal corresponde aos mecanismos usuais de UNIX e C para definição de constantes de sistema.

```
/* função chamada pela concurrent_call */

func(nc, arg_1, ..., arg_n)
int nc ;          /* número da chamada */
...
{
        código da função ;
}
```

Figura 3.2: Função executada concurrentemente por tarefas criadas pela primitiva *concurrent_call*

- se *"call_flag"* = *"CNCALL_NUMPROC"* : o sistema cria uma tarefa para cada processador configurado como pertencendo ao complexo. Neste caso, o argumento "n" (número de activações da função chamada em concorrência) deve ser omitido, uma vez que será igual ao número de unidades de processamento que formam o complexo.

- se *"call_flag"* = *"CNCALL_INFINITE"* : o sistema cria continuamente tarefas, cada uma executando o código de "func". É da responsabilidade do programador providenciar mecanismos para sair explicitamente deste ciclo infinito e assegurar-se que o programa eventualmente terminará.

Uma **concurrent_call()** pode terminar explicitamente ou implicitamente através de uma escolha apropriada da variável *"quit_flag"*:

- se *"quit_flag"* = *"CNCALL_NO_QUIT"* a execução concorrente terminará apenas depois de todas as tarefas terem sido criadas, terem executado e terminado.

- se *"quit_flag"* = *"CNCALL_NONZERO_QUIT"* ou se *"quit_flag"* = *"CNCALL_NEGATIVE_QUIT"* a execução concorrente terminará desde que uma das tarefas devolva um valor não nulo (ou negativo). Esse valor é passado como valor de retorno da *concurrent_call()*,

Estão igualmente disponíveis primitivas que permitem evitar ou possibilitar que o complexo seja desmembrado num dado instante de tempo. Tais primitivas, *quit_enable()*

e *quit_disable()*, devem rodear código que o programador sabe não dever ser interrompido e o sistema assegura que mesmo que termine a fatia de tempo assignada à composição do complexo, este não é desmembrado sem o código ter terminado de se executar. Em funcionamento normal, no entanto, a *concurrent_call()* espera que todas as tarefas (ou apenas as já criadas) terminem, para prosseguir, o que corresponde de certa forma a uma espécie de primitiva de sincronização no final.

Duas outras primitivas de uma biblioteca do sistema ("common library") são de interesse geral e particularmente no contexto deste trabalho:

lib_number_of_processors()

que devolve o número de **CE** definido como sendo parte do complexo computacional (note-se que isto permite uma certa flexibilidade de programação, pois possibilita o desenvolvimento de programas independentes da configuração particular da máquina), e

lib_processor_number()

que devolve o número do processador em que a tarefa está a ser executada. Ambas as funções devem ser usadas apenas em modo concorrente, ou seja, devem ser chamadas apenas no contexto de funções executadas em concorrência (caso contrário o valor devolvido é 0).

O sistema operativo do *Alliant* oferece igualmente mecanismos tradicionais para partilha de memória entre processos. No entanto, para aplicações baseadas em paralelismo de tarefas, estes mecanismos não são de interesse particular, dado que todas as tarefas criadas pela primitiva *concurrent_call()* coexistem no contexto do processo que as criou e herdam acesso directo a todo o seu espaço de endereçamento. A possibilidade de definir zonas de memória comuns a vários processos pode no entanto ser de interesse para certo tipo de aplicações.

Numa máquina com as características do *Alliant* é essencial a existência de mecanismos de sincronização. **Semáforos atómicos** para permitirem fechar ou abrir o acesso a certas zonas de memória, podem ser usados por um conjunto de tarefas ou processos partilhando o acesso a dados em memória. O *Alliant*, cujo sistema operativo é baseado na versão **UNIX 4.3bsd** da Universidade da Califórnia não contém os semáforos habituais do sistema UNIX, oferecendo em contrapartida as funções *lock()* e *unlock()*, com funcionalidade semelhante. Estas funções formam a base dos mecanismos de sincronização entre tarefas na máquina, dado que qualquer outro mecanismo (utilização dos sinais do

UNIX[13], passagem de mensagens, etc) são menos naturais, atendendo a que as tarefas coexistem no espaço de um simples processo e partilham todo o espaço de memória deste.

Alguns testes de avaliação de desempenho foram conduzidos no *Alliant*. Estes focaram avaliar o desempenho da máquina em tarefas rotineiras como: criação de tarefas concorrentes, mecanismos de sincronização, criação de processos, etc. O facto de que a criação de tarefas num ambiente concorrente é um mecanismo extremamente rápido (cerca de 400 vezes mais rápido que criar um processo novo com a função de sistema "*fork()*") é um bom indicador do nível de desempenho que pode ser conseguido nesta máquina. A tabela 3.1 contém o conjunto de dados mais interessantes dentre as medidas efectuadas.

Primitiva	Tempo de Execução (segundos)
fork()	0.8 m
concurrent_call()	2 μ
lock() or unlock()	40 μ

Tabela 3.1: Testes de Desempenho num Alliant FX

As implementações descritas nesta tese foram feitas utilizando o mecanismo de criação de tarefas concorrentes, com *concurrent_call()*, e sincronizando o acesso dessas tarefas a dados comuns através da utilização de semáforos. Cada conjunto de dados partilhados tem assim associado um semáforo, sobre o qual a aplicação de *lock()* ou *unlock()* garante a exclusividade de acesso.

[13] Através da primitiva *signal()*.

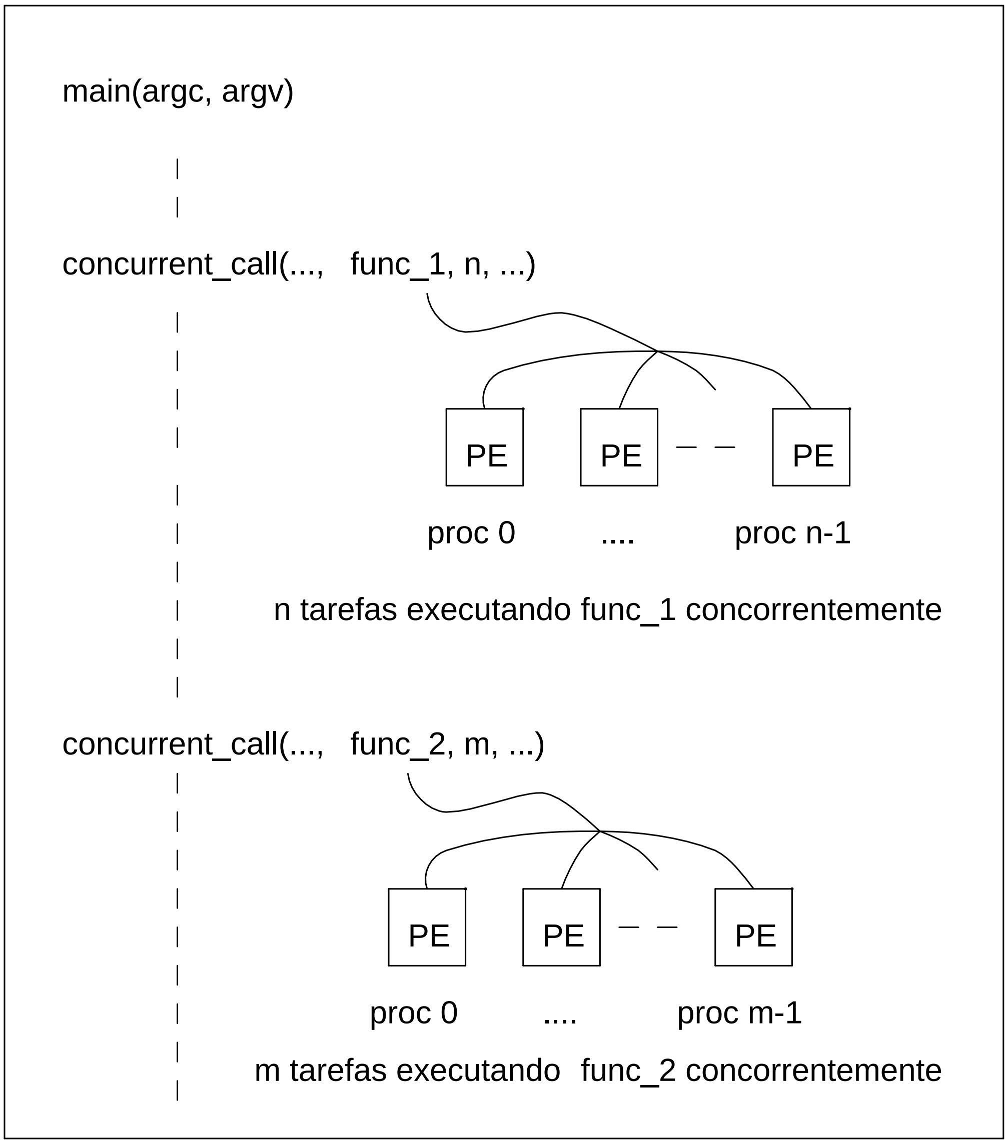

Figura 3.3: Paralelismo por criação de tarefas concorrentes num Alliant

Capítulo 4

Implementações em Multiprocessador

Este capítulo descreve a arquitectura e as opções tomadas no desenvolvimento de três versões de um simulador de circuitos ao nível eléctrico. Cada versão foca um aspecto diferente da procura de paralelismo ao nível das estruturas ou funcionalidade de um algoritmo, baseando-se num determinado conjunto de opções de projecto. O estudo e desenvolvimento destas três versões foi efectuado no intuito de testar os vários tipos de paralelismo exploráveis num algoritmo de simulação ao nível eléctrico, tendo estas sido posteriormente implementadas numa sequência que se afasta progresivamente do algoritmo original de **CINNAMON**. Os protótipos desenvolvidos destinam-se a ser implementados sobre computadores explorando uma arquitectura paralela, possibilitando uma aceleração real do processo de simulação [Silveira87ii].

Qualquer das versões foi implementada sobre um computador **Alliant**, um multiprocessador utilizando uma arquitectura centralizada, baseada numa memória comum. Embora a arquitectura desta máquina, descrita na secção 3.4, seja bastante rígida, ela pode de certa maneira emular outro tipo de máquinas, o que será abordado adiante.

As versões do simulador apresentadas baseiam-se no algoritmo do simulador **CINNAMON**. No entanto, e como veremos, afastam-se dele de forma cada vez mais acentuada, por razões relacionadas com a arquitectura da máquina ideal a que cada versão se destina. Esta será, em cada caso, descrita de forma geral, procurando-se explicar a motivação para a sua escolha.

Por detrás de qualquer implementação de um algoritmo de CAD numa arquitectura paralela, existem um número de **opções de desenho** que têm necessariamente de ser tomadas. Tais opções estão relacionadas com o paralelismo explorado ao nível dos algoritmos, com a estratégia seguida no seu desenvolvimento e com a arquitectura do processador a que se destinam.

De uma forma geral este problema pode ser olhado do ponto de vista **algorítmico** ou **arquitectónico** [Lorin72]. Do ponto de vista algorítmico, tais implementações resultam da procura de paralelismo inerente na estrutura de um dado problema, podendo ser adaptações de algoritmos sequenciais ou resultar do desenvolvimento de novos algoritmos que exploram de forma mais eficiente o paralelismo inerente ao problema que se pretende resolver. Do ponto de vista arquitectónico, elas permitem-nos definir a arquitectura de uma máquina que explore, da forma mais eficaz, as potencialidades desse algoritmo.

Em geral, a interacção construtiva dos dois tipos de aproximações mencionados permite conduzir a estruturas que satisfazem o objectivo último da implementação: o de permitir a resolução mais rápida de um dado problema ou de problemas semelhantes, mas de maiores dimensões.

Nas secções que se seguem procura-se explicar de que forma esse processo foi abordado e os resultados daí obtidos, começando por se estabelecer um conjunto de definições relativas ao ambiente de trabalho.

4.1 Definições e Ambiente de Desenvolvimento

Antes de iniciar a descrição das versões desenvolvidas no decorrer deste trabalho, é importante começar por definir um conjunto de regras e conceitos sobre os quais assentará a análise dos resultados obtidos.

A implementação paralela de um algoritmo num computador com n processadores possibilita, no caso ideal, a obtenção de um factor de aceleração de n, sobre uma implementação sequencial. Na prática, no entanto, apenas esporadicamente isso é possível devido a um conjunto de factores que afectam de forma decisiva o desempenho observado. Basicamente como causas dessa diminuição, podemos destacar dois pontos fundamentais [Sadayappan]:

- a computação que é necessário efectuar sem directa ligação com o algoritmo em causa *(execution overheads.)*.
- a existência de processadores em estados de espera durante a execução do algoritmo *(processor idling.)*.

De facto, na execução de um algoritmo paralelo, um processador dispende um determinado esforço na realização de tarefas não directamente ligadas à execução do algoritmo em si. Entre as causas para este *overhead* podemos distinguir:

- *overhead* na representação e modificação do código: as alterações introduzidas num algoritmo de forma a facilitar a exploração de paralelismo podem resultar num aumento de *overhead*, seja na partição dos dados do problema, seja na manutenção da consistência dos mesmos, etc.

- *overhead* devido à execução de operações concorrentes: a exploração de paralelismo numa implementação num multiprocessador é feita através da execução concorrente de tarefas em vários processadores. A alocação, sincronização e conclusão dessas tarefas, feita recorrendo a operações concorrentes específicas da máquina, implicam uma diminuição do desempenho observado, que não é possível, no entanto, evitar.

- *overhead* vectorial: a execução de operações vectoriais, em máquinas com essa capacidade, embora possibilitando a obtenção de um melhor desempenho, implica um determinado esforço na iniciação de tais operações. De facto, é necessária a colocação dos dados a operar, nos registos vectoriais, o que a ser feito por endereçamento indirecto pode diminuir consideravelmente o desempenho observado[1].

Aos factores apresentados anteriormente, acresce a diminuição de desempenho observada devido à eficiência de utilização dos processadores ser reduzida pelo facto de em diversos pontos do algoritmo ser necessário ter processadores em estados de espera. As causas fundamentais para estes estados de espera são em geral:

- desiquilíbrio na distribuição da carga computacional entre os vários processadores *(load imbalance)*. Um caso importante deste problema é capturado pela conhecida **Lei de Amdahl**. Esta relaciona-se com a situação, natural, em que uma determinada fracção do código é completamente sequencial, ou seja, a distribuição de carga é tão desigual que apenas um processador está a executar um trabalho útil, estando os outros num estado de espera. Se uma fracção f do código tem de ser executada sequencialmente, mesmo que o restante possa ser paralelizado de forma perfeita, a aceleração máxima que é possível obter numa máquina com n processadores, definida como a razão entre os tempos de execução entre versões paralela e

[1]Este ponto não é no entanto de grande interesse no contexto desta tese, uma vez que não foram utilizados os mecanismos de computação vectorial disponíveis no *Alliant*.

sequencial, é limitada por[2]:

$$\eta = \frac{n}{n\, f + 1 - f} \tag{4.1}$$

Note-se as implicações desta lei, que indicam, por exemplo, que com 16 processadores, se 20% do código tiver de ser executado sequencialmente, a aceleração máxima observável é apenas de 4, ou que se o número de processadores tende para infinito (infinidades de recursos computacionais) a aceleração máxima tende para $1/f$.

- contenção no acesso a recursos partilhados *(resource contention)*. A necessidade de partilhar recursos implica a sincronização no acesso aos mesmos por parte dos vários processadores. Como exemplos desta contenção temos a existência de memória ou canais de entrada-saída partilhados. Acresce que essa sincronização é geralmente feita através de semáforos que os vários processadores têm de estar continuamente a observar, o que aumenta os potenciais conflitos no acesso a memória.

A importância relativa dos vários factores descritos será claramente dependente do algoritmo e da máquina em causa. De forma geral, no entanto, é possível reconhecer a existência de um compromisso entre o desiquilíbrio da carga computacional e os diversos *overheads*, dependendo da granularidade do algoritmo. Algoritmos de maior granularidade diminuem os *overheads* mas podem tornar o equilíbrio da carga entre os vários processadores extremamente difícil de manter. Ao contrário, algoritmos de grão mais fino reduzem este último problema à custa de um aumento de *overhead* (por exemplo de comunicação).

Torna-se claro neste momento que a aceleração conseguida por um algoritmo paralelo será dependente da forma como o mesmo se adaptar à arquitectura sobre a qual é implementado. De facto, alguns dos factores descritos anteriormente são desprezáveis em certo tipo de arquitecturas, podendo no entanto ser preponderantes noutras.

Existe uma variação considerável na literatura sobre o processo de medição da razão de aceleração conseguida através de um algoritmo paralelo. Quantidades como eficiência da paralelização, percentagem de código executada em paralelo, etc, têm sido introduzidas como padrões do desempenho e utilizadas por vários autores. A aceleração, ou eficiência de paralelização, introduzida anteriormente (ver equação 4.1), é geralmente considerada a medida mais correcta do desempenho de uma implementação paralela.

[2]Note-se que se trata de uma relação matematicamente precisa e não de uma estimativa.

Definição 4.1 (Aceleração ou Eficiência de Paralelização)

A Aceleração ou Eficiência de Paralelização de um dado algoritmo é definida como o quociente dos tempos de execução de uma versão paralela e da correspondente versão sequencial do mesmo algoritmo. Esta relação será designada por η.

Atendendo no entanto a que os protótipos aqui descritos foram desenvolvidos sobre uma máquina com uma arquitectura que não é necessariamente aquela que melhor se ajusta a cada algoritmo, parece-nos incorrecto utilizar o factor de aceleração, obtido apenas pela observação dos tempo de execução das diversas implementações efectuadas face à versão sequencial, como medida absoluta de desempenho de cada algoritmo. Será portanto necessário ter em conta as características específicas do *Alliant* e contrapô-las com a arquitectura considerada ideal em cada caso. O ponto essencial desta distinção assenta na escolha da versão sequencial para a qual aquele quociente é calculado. Em geral, utilizar-se-ia como padrão de base os resultados da execução da versão sequencial de algoritmo original de **CINNAMON**, considerando-se no entanto, em certos casos, que tal não é o processo mais correcto de efectuar a comparação pretendida. Assim, para cada protótipo será motivado o processo considerado mais correcto de determinar a aceleração com ele conseguido e a mesma será calculada.

4.2 Uma Versão Acelerada

Uma primeira implementação do algoritmo de base de **CINNAMON** foi levada a cabo no sentido de testar as potencialidades oferecidas pelo *Alliant* para exploração do paralelismo existente no algoritmo de simulação. Com esta primeira versão foi estudada a arquitectura da máquina, as potencialidades que oferece a um programador e a forma como as mesmas podem ser eficientemente aproveitadas.

De uma certa forma esta primeira versão não deve ser considerada uma versão paralela, sendo mais correctamente uma versão acelerada. A razão para esta diferenciação é discutível mas assenta fundamentalmente no facto de o algoritmo de base desta versão não ser um algoritmo paralelo em si, mas directamente derivado do algoritmo de base da simulação.

4.2.1 Algoritmo de Controlo

O algoritmo de controlo da simulação em **CINNAMON**, apresentado anteriormente (Algoritmo 3.2), foi essencialmente mantido nesta versão, tendo sido explorado o paralelismo de dados a ele inerente. Com base numa aproximação de grão muito fino, começou por se efectuar uma partição funcional do algoritmo, tentando-se posteriormente determinar quais os conjuntos de acções que poderiam ser executados sobre vários dados em simultâneo.

Uma observação cuidadosa do algoritmo apresentado indica imediatamente a existência de três tipos de acções necessárias à resolução de um acontecimento num nó.

Em primeiro lugar, uma fase necessariamente sequencial, que se relaciona com a obtenção da informação relativa ao acontecimento a ser processado e posteriormente com o arquivo dos resultados desse processamento. Em segundo lugar uma fase associada à computação dos coeficientes necessários a essa resolução. Finalmente, uma terceira fase que se relaciona directamente com os cálculos necessários para a resolução do acontecimento e que pode causar a marcação de novos acontecimentos.

Estas três fases do ciclo de simulação foram consideradas de forma diversa no desenvolvimento desta versão. A primeira fase, dado ser sequencial, foi deixada inalterada dado que pouco haveria a ganhar com alterações na sua estrutura[3]. As duas restantes fases merecem no entanto uma atenção mais cuidada.

Atendendo a que o algoritmo de **CINNAMON** desacopla em cada instante as equações do circuito, a determinação dos coeficientes da equação diferencial linear (equação 3.12) associada a cada acontecimento implica em cada instante a determinação da contribuição apenas de um pequeno grupo de elementos do circuito. Mesmo este processo simplificado origina no entanto a linearização de todos os elementos do circuito ligados ao nó que causou o acontecimento. No caso dos dispositivos semi-condutores, tal implica a computação de um conjunto de parâmetros envolvendo um grande número de operações elementares[4]. Note-se no entanto que a linearização dos dispositivos e o cálculo das suas contribuições para a equação do nó em causa é essencialmente uma tarefa sequencial para cada dispositivo. De facto esse processo necessita apenas da informação sobre o estado dos nós ligados ao dispositivo, dados esses que são globais e constantes durante o processo de linearização. Isto indica que os vários elementos podem ser linearizados

[3] Convém no entanto referir que o processo de arquivo de resultados pode ser responsável por algo como 30% do tempo de simulação observado.

[4] O estudo dos resultados de simulações efectuadas com CINNAMON indicam que o tempo de cálculo dos coeficientes da equação diferencial ronda em média 1/3 do tempo total de simulação.

independentemente e logo em simultâneo. Desta forma se depreende que a exploração de paralelismo a este nível pode, com um esforço mínimo (dado não haver qualquer tipo de dependência entre dados), conduzir a aceleração da execução do algoritmo.

A resolução de um acontecimento em **CINNAMON** está como se sabe relacionada com alterações no estado de um nó, designado como activo. Essas alterações, na tensão do nó, correspondem a perturbações que podem levar a futuras alterações no estado do próprio nó e de nós vizinhos. Tais perturbações propagam-se na vizinhança do nó activo, podendo conduzir a alterações no estado de todo o circuito. Essencialmente isso significa que a resolução de um acontecimento obriga a recalcular o nó activo e aqueles que a ele estão ligados, de maneira a determinar de que forma essa perturbação se processa. De novo, e atendendo a que se trata de um algoritmo explícito, esse processamento pode ser feito em paralelo, levando assim a uma redução no tempo total de simulação.

Existe no entanto uma diferença significativa entre as duas fases apresentadas anteriormente como paralelizáveis. De facto, a possibilidade de uma perturbação num nó se poder propagar à vizinhança desse nó implica que um ou mais nós podem necessitar de ser remarcados como resultados dessa propagação. Atendendo a que a lista de acontecimentos para o processo de simulação é única, isto implica, potencialmente, que diferentes processadores podem tentar alterar o estado dessa lista simultaneamente.

De forma a garantir a consistência dos dados, é então necessária uma metodologia que force a sincronização no acesso à lista de acontecimentos. Na implementação aqui discutida, isso foi conseguido pela utilização de um semáforo associado à lista de acontecimentos. Quando um processador pretende alterar a estrutura da lista, começa por requerer o acesso à mesma. Se tal lhe é concedido, então esse acesso é exclusivo, pelo que a consistência dos dados é garantida entre cada alteração[5]. Caso a lista esteja a ser alterada por outro processador, o processador requerente esperará até que lhe seja concedido o acesso.

Em termos gerais, o algoritmo de controlo da simulação, para a versão acelerada é o seguinte[6]:

[5]Não é no entanto garantida a sequência de alterações devidas aos vários nós que são recalculados em simultâneo. Tal pode levar a pequenas diferenças nos resultados de simulação, o que em primeira aproximação é desprezável.

[6]O apêndice C consiste numa versão um pouco mais detalhada do algoritmo.

Algoritmo 4.1 (Cinnamon: Versão Acelerada)

```
simulação()            /* seja DC ou Transiente */
{
    t = 0 ;
    marcação das fontes de tensão;
    enquanto( há acontecimentos na lista e t < TempoMáximo) {
        lê a informação do próximo acontecimento (nó, tempo, tensão) ;
        t = tempo ;
        disp = número de dispositivos ligados ao nó activo ;
        /* em geral um nó é considerado vizinho dele próprio */
        viz = número de nós adjacentes ao nó activo;
        concurrent_call(..., lineariza dispositivos, disp, nó) ;
        concurrent_call(..., propaga perturbação, viz, nó) ;
        guarda informação do acontecimento (nó, tempo, tensão) ;
    }
}
```

Para possibilitar uma maior aceleração do processo de simulação, foi utilizada posteriormente uma outra técnica. Através do estudo cuidadoso de perfis dos tempos de simulação[7] foi possível isolar outras partes do código computacionalmente mais intensivas, nomeadamente ciclos, que foram em seguida paralelizadas. Esta técnica originou uma diminuição muito ligeira do tempo de simulação, tendo fundamentalmente sido explorada no sentido de permitir um contacto mais profundo com os mecanismos de concorrência disponíveis no sistema.

Este primeiro protótipo do simulador **CINNAMON** foi, como os restantes, implementado no *Alliant* existente no INESC. Nos capítulos seguintes são apresentados e discutidos os resultados obtidos e as conclusões retiradas deste primeiro trabalho. Nesta versão acelerada foi unicamente explorado o paralelismo ao nível dos dados, e apenas em partes bem determinadas do código. Se bem que exista um paralelismo bem evidente ao nível dos dados, nesta versão não se avaliam acontecimentos em simultâneo, efectuando-se em paralelo apenas um conjunto de cálculos relacionados com a resolução de cada acontecimento. Desta forma apenas marginalmente se pode falar em paralelismo de dados.

[7]Para tal foi utilizada uma ferramenta UNIX que permite obter perfis dos tempos de execução de um programa: *prof*.

É interessante registar algumas das características destas versão que apesar da sua simplicidade a tornam interessante do ponto de vista de algoritmo paralelo. Por um lado, note-se que o paralelismo é explorado na resolução de cada acontecimento, pelo que automaticamente se efectua uma partição dinâmica dos dados, ditada pela própria actividade do circuito e descrita pela sequência presente na lista de acontecimentos. Esta característica é herdada da própria estrutura de controlo de simulação por acontecimentos, atendendo a que ela se baseia na exploração da latência existente num circuito. Por outro lado sugere de certa forma a existência de uma partição funcional ao nível do próprio algoritmo, uma vez que permite separar de forma clara uma sequência de acções cuja execução é necessária para a resolução de cada acontecimento. De facto, ao explorar-se paralelismo na fase de linearização e posteriormente na determinação dos coeficientes da equação diferencial e sua resolução, sugere-se implicitamente a possibilidade de construção de um *pipeline* sobre o algoritmo original. Tal não pode, infelizmente, ser colocado de forma assim tão simplista, atendendo a que como se sabe cada acontecimento pode implicar uma alteração completa da ordenação da lista de acontecimentos, o que acarretaria um esvaziamento da *pipeline*. Essa situação deve aliás considerar-se como bastante vulgar e acarreta um *overhead* considerável no processo de simulação.

4.2.2 Considerações Arquitectónicas

Em termos arquitectónicos, a versão descrita na secção anterior apresenta características que restringem a sua utilização a máquinas de memória partilhada, onde a criação e sincronização de tarefas seja um processo computacionalmente pouco pesado. Não significa isto que esta versão seja apenas utilizável no *Alliant*, dado que como foi referido, grande parte das máquinas comerciais disponíveis partilham várias das suas características arquitectónicas, nomeadamente a estrutura de memória partilhada. Não é no entanto indicada para uma arquitectura descentralizada, situação que aliás se generaliza, como foi visto, para qualquer algoritmo de simulação controlada por acontecimentos.

É importante notar igualmente que, apesar de se tratar de uma versão explorando paralelismo de grão fino, se destina a máquinas com um reduzido número de processadores. De facto, repare-se que tendo o paralelismo sido explorado ao nível da linearização simultânea de dispositivos e da reavaliação simultânea de nós isso restringe o número de processadores potencialmente activos ao mesmo tempo. No primeiro caso, a aceleração máxima atingível é limitada pelo número médio de dispositivos ligados a cada nó, enquanto que no segundo caso a conectividade média de um nó estabelece igualmente um limite superior à aceleração que se poderá obter. Como em circuitos típicos qualquer destes números é em geral inferior a 6, tal estabelece um limite máximo ao número de processadores utilizáveis em simultâneo. É portanto claro que a aceleração máxima que se pode obter com este algoritmo não é escalável com o número de processadores, sendo dependente de forma precisa, das características do circuito a ser simulado.

4.3 Paralelismo por Discretização Temporal

A versão acelerada descrita na secção anterior pode ser vista como assentando numa aproximação de grão fino baseada numa partição de dados efectuada de forma dinâmica. Como desvantagens dessa versão foi fundamentalmente apontada a sua reduzida expansibilidade a máquinas com um número elevado de processadores. Nesta secção descreve-se um algoritmo paralelo, derivado do tipo de aproximações explorado em **CINNAMON**, implementado no *Alliant*, mas fundamentalmente indicado para arquitecturas com um elevado número de processadores, como as máquinas massivamente paralelas apresentadas na secção 2.1.2.

O objectivo do desenvolvimento deste protótipo é o de testar o paralelismo inerente a uma aproximação explícita do tipo exponencial, através de um algoritmo explorando paralelismo de grão fino. Como se verá, o algoritmo implementado afasta-se bastante do algoritmo de **CINNAMON**, apesar de aqui se designar como uma implementação paralela desse simulador.

A exploração de paralelismo por discretização temporal aparece no seguimento do trabalho desenvolvido na versão acelerada e tenta resolver algumas das desvantagens que aquela apresentava. Trata-se no entanto de um algoritmo bastante mais geral do que aquele, sobretudo no que respeita ao processo de controlo da simulação, dado este ser efectuado de forma verdadeiramente distribuída.

4.3.1 Discretização Temporal e Partição de Dados

Para motivar o algoritmo descrito nesta secção, reportemo-nos à versão acelerada, descrita na secção anterior, e procure-se encontrar formas de evitar as desvantagens que lhe foram apontadas. Por um lado acentou-se o facto de se tratar de uma versão em que o paralelismo apenas é explorado ao nível das operações elementares envolvidas na resolução de cada acontecimento. A obtenção de uma maior aceleração do processo de simulação passa então, de alguma forma, pela resolução simultânea de vários acontecimentos em vários nós do circuito.

Foi igualmente apontado na secção anterior o facto do factor de aceleração obtido com a versão acelerada não ser proporcional ao número de processadores disponíveis, uma vez que o paralelismo máximo obtido com aquela versão está limitado por aspectos da topologia do circuito. Uma vez que aqui se pretende igualmente explorar paralelismo de grão fino, dever-se-á da mesma forma partir de uma estrutura simples, e explorar paralelismo ao nível da resolução de um número elevado de problemas de pequena dimensão, mas evitando qualquer dependência com a topologia do circuito.

Uma possível solução para este último problema consiste em considerar cada nó como uma estrutura de dados elementar e obter paralelismo pela determinação simultânea da evolução da dinâmica de cada nó. Uma vez que se pretendem simular circuitos de grandes dimensões, isto possibilita a utilização de máquinas com um grande número de unidades computacionais e, potencialmente, a obtenção de elevada aceleração[8].

A questão fundamental neste caso é de como explorar este paralelismo ao nível de cada nó. Uma possível resposta foi já mencionada e consiste na resolução simultânea de vários acontecimentos relativos a vários nós. Este processamento simultâneo de vários acontecimentos não é no entanto tão claro como pode parecer, uma vez que, como foi já debatido (secção 3.3.4), a sequência dos acontecimentos na respectiva lista, pode ser alterada a qualquer momento pela resolução de um qualquer dentre eles.

Na verdade, por definição, a lista de acontecimentos consiste numa sequência de acontecimentos correspondentes à propagação de perturbações num circuito. A resolução de cada acontecimento pode implicar a remarcação ou desmarcação de todos ou parte dos restantes acontecimentos na lista. Desta forma, não é possível num dado instante saber se acontecimentos marcados para instantes no futuro estarão ainda na lista na altura apropriada para a sua resolução.

A aproximação tentada nesta versão consiste no abandono do processo de controlo da simulação por acontecimentos e na utilização de técnicas usuais de discretização do eixo dos tempos. Esta metodologia permite a exploração de paralelismo em cada instante, na determinação da evolução da tensão de cada nó do circuito. Este tipo de metodologia é de certa forma semelhante a outra tentativas realizadas no contexto de simulação ao nível lógico e apropriadas para implementação em arquitecturas massivamente paralelas [Denneau82].

[8] Repare-se que se o número de processadores é muito elevado, mesmo uma aplicação apresentando uma eficiência de paralelização mínima pode conduzir a uma aceleração importante na resolução de um dado problema.

A utilização de técnicas de discretização temporal requer, como visto na secção 3.2, a implementação de um algoritmo numérico de integração do sistema de equações diferenciais que descreve o comportamento do circuito. No contexto da metodologia do simulador **CINNAMON**, foi implementado um algoritmo explícito por ajuste de exponencial que permite, sob certas condições, o desacoplamento total das equações do sistema. Ao contrário da metodologia encetada em **CINNAMON**, a equação resultante para cada nó, é agora resolvida de forma a obter um novo valor de tensão, dependente do passo de integração utilizado.

Partindo então da equação resultante, para um nó k genérico e admitindo como correcto (dentro dos constragimentos enunciados) o desacoplamento das equações do circuito:

$$c_{kk} \frac{d}{dt}\hat{v}_k(t) + g_{kk}\ \hat{v}_k(t) = \hat{\imath}_k(t) \tag{4.2}$$

obtem-se então:

$$\hat{v}_k(t + h_k^{(t)}) = (\hat{v}_{ko} - \hat{v}_{k\infty})e^{(h_k^{(t)})/\tau_k} + \hat{v}_{k\infty} \tag{4.3}$$

em que $h_k^{(t)}$ é o passo de integração utilizado no instante t para o nó k.

A partição dos dados de um problema é uma técnica derivada de uma estratégia de *"dividir para conquistar"*. A ideia essencial consiste na divisão de um problema de grande dimensão num conjunto de problemas menores, que serão então resolvidos. Em simulação, a partição de um circuito é uma técnica muito utilizada, atendendo à grande quantidade de dados envolvidos no problema [Cox86] [Agrawal85]. A metodologia indicada anteriormente estabelece um processo automático e eficiente de partição do dados de um problema, ao mesmo tempo que estabelece de forma clara como esses dados devem ser tratados individualmente.

De facto, é interessante salientar que a utilização de técnicas de discretização do eixo dos tempos associadas a um algoritmo de integração numérica explícito, proporciona uma forma de partição ao nível dos dados do problema, ao permitir o desacoplamento total das equações do sistema em cada instante.

A estratégia de partição de dados utilizada no algoritmo implementado é assim inerente à própria exploração de paralelismo e proporciona a divisão imediata do circuito num conjunto de entidades com estrutura própria: os nós. Uma outra aproximação ao problema, apresentada recentemente [Ghosh87], tem vários pontos em comum com a metodologia aqui descrita. Nela modela-se cada componente, um nó no nosso caso, como uma **Descrição Executável**, referida como um **Modelo**, que é uma entidade inteligente no sentido em que executa tarefas de marcação própria, simulação e verificação. Um modelo marca-se a si próprio para execução quando estão reunidas as condições necessárias nos seus terminais de entrada. Uma lista de dispositivos ligados a cada terminal de saída é mantida e as perturbações são propagadas para cada um dos componentes quando necessário. No nosso caso a discretização temporal evita grande parte desta computação, dado que todos os nós serão calculados em cada instante.

4.3.2 Algoritmo de Controlo

Na secção anterior foi descrita a metodologia seguida no desenvolvimento de um algoritmo paralelo de simulação eléctrica de circuitos baseado numa aproximação de grão fino e utilizando técnicas de discretização do eixo dos tempos. De acordo com a metodologia proposta, é agora suficiente definir um processo de determinação do passo de integração a usar para cada nó, em cada instante.

No algoritmo implementado, utiliza-se um esquema denominado de **passo variável**, no qual o passo de integração não é fixo, sendo em cada instante determinado de acordo com a dinâmica do circuito. No entanto, de forma a manter o sincronismo entre as várias entidades de simulação, os nós, é utilizado em cada instante o mesmo passo de integração para todos os nós.

O processo utilizado para determinação do passo de integração a utilizar em cada instante deriva das técnicas de controlo da simulação utilizadas em **CINNAMON**. Este inicia-se calculando, para cada nó, qual o intervalo de tempo necessário até a sua tensão se alterar de $\pm\Delta V$. Este procedimento foi sugerido através do próprio algoritmo de **CINNAMON** e é extremamente interessante uma vez que implica um paralelismo total ao nível dos dados. Uma vez determinado, para cada nó, o instante em que ele sofrerá uma alteração, é então determinado o mínimo dentre esses tempos, que é utilizado como passo de integração.

Desta forma, calcula-se, para cada nó k do circuito:

$$h_k^{(t)} = -\tau_k \ln\left(\frac{\hat{v}_{ko} - \hat{v}_{k\infty} \pm \Delta V}{\hat{v}_{ko} - \hat{v}_{k\infty}}\right) \tag{4.4}$$

e escolhendo-se depois

$$h^{(t)} = min\{h_k^{(t)}, \ \forall k\} \tag{4.5}$$

como passo de integração a utilizar no instante t.

Em termos gerais, o algoritmo de controlo da simulação, para a versão sequencial, é o seguinte.

Algoritmo 4.2 (Efit: Versão Sequencial)

```
simulação()              /* seja DC ou Transiente */
{
    t = 0;
    marcação das fontes de tensão;
    enquanto( o circuito não atingir um estado estável e t < TempoMáximo)
      {
        lineariza todos os dispositivos do circuito;
        determina o intervalo de tempo antes da próxima alteração para
          cada nó;
        h = mínimo dentre os intervalos determinados;
        determina a variação da tensão em cada nó(t, h);
        t = t + h;
        guarda informação do estado do circuito(tempo);
    }
}
```

Note-se que da forma descrita, não é necessária a manutenção de uma lista de acontecimentos nos nós. Por razões de comodidade será no entanto mantida uma estrutura de dados, que se denominará lista de actividade, e que conterá para cada nó uma indicação da sua evolução, bem como, em cada instante, o intervalo de tempo que esse nó necessita para atingir uma variação de $\pm\Delta V$ no valor da sua tensão.

A exploração de paralelismo no algoritmo anterior é agora perfeitamente óbvia, uma vez que as três fases computacionalmente mais intensivas, de linearização dos dispositivos, determinação de uma estimativa do passo de integração para cada nó e cálculo da evolução da tensão em cada nó, podem ser efectuadas em simultâneo. De facto, a linearização dos dispositivos é um processo que pode ser conduzido em paralelo para todos os elementos de circuito, tal como o cálculo do passo de integração e posterior determinação da variação de tensão em cada nó, uma vez que a utilização dum método explícito permite um desacoplamento total das equações relativas a cada nó.

Em termos gerais, o algoritmo de controlo da simulação, para a versão paralela, é o seguinte[9]:

Algoritmo 4.3 (Efit: Versão Paralela)

```
simulação()              /* seja DC ou Transiente */
{
    t = 0;
    marcação das fontes de tensão;
    enquanto( o circuito não atingir um estado estável e t < TempoMáximo)
      {
        disp = número de dispositivos presentes no circuito;
        nós = número de nós do circuito;
        concurrent_call(..., lineariza dispositivos, disp);
        concurrent_call(..., determina intervalo de propagação de per-
          turbação, nós);
        h = mínimo dentre os intervalos determinados;
        concurrent_call(..., determina a variação da tensão, nós, h);
        t = t + h;
        guarda informação do estado do circuito(tempo);
    }
}
```

4.3.3 Considerações Arquitectónicas

O algoritmo paralelo descrito anteriormente e ao contrário do simulador **CINNAMON**, não procura explorar a latência existente num circuito electrónico. De facto, de acordo com a metodologia proposta, em cada instante no eixo dos tempos todo o circuito será reavaliado, o que requer a existência de um elevado número de processadores, sob pena de se tornar ineficaz o processo de simulação.

Este algoritmo destina-se portanto prioritariamente a máquinas massivamente paralelas com um elevado número de unidades funcionais, de forma a que cada uma, basicamente, se ocupe da determinação da evolução da dinâmica de cada nó do circuito. Numa

[9] O apêndice D consiste numa versão um pouco mais detalhada do algoritmo.

primeira aproximação este algoritmo pode ser testado numa máquina com a arquitectura do *Alliant*, estando no entanto neste caso a aceleração observada, limitada pelo número de unidades de processamento disponíveis, enquanto que se pretende que essa aceleração seja apenas limitada pela dimensão do circuito.

Como exemplo de uma máquina cuja arquitectura se adapta de forma perfeita às características do algoritmo descrito, temos uma matriz de processadores, apresentada na figura 2.4, ou uma máquina como a *Connection Machine*, cuja arquitectura é apresentada na figura 4.1.

4.4 Paralelismo por Partição de Dados

As versões descritas nas secções anteriores, embora algoritmicamente bastante distintas, partilham no entanto uma característica muito importante. De facto, ambas foram desenvolvidas no sentido de explorar um paralelismo de dados de grão muito fino, baseando-se para tal na estrutura de dados mais elementar do simulador, o nó do circuito, e procurando paralelismo ao nível de operações elementares.

Nesta secção descrevem-se as opções tomadas no desenvolvimento de uma versão baseada em paralelismo de dados, mas explorando uma arquitectura de simulação de grão grosso. Embora apoiando-se no algoritmo de base de **CINNAMON**, estende os conceitos nele desenvolvidos num sentido hierárquico, nomeadamente no aspecto das estruturas de dados utilizadas e no controlo da simulação. Esta versão afasta-se da versão descrita na secção 4.3, não só em termos de granularidade do algoritmo, mas igualmente porque retorna às técnicas de discretização do eixo das tensões, características da aproximação explorada em **CINNAMON**.

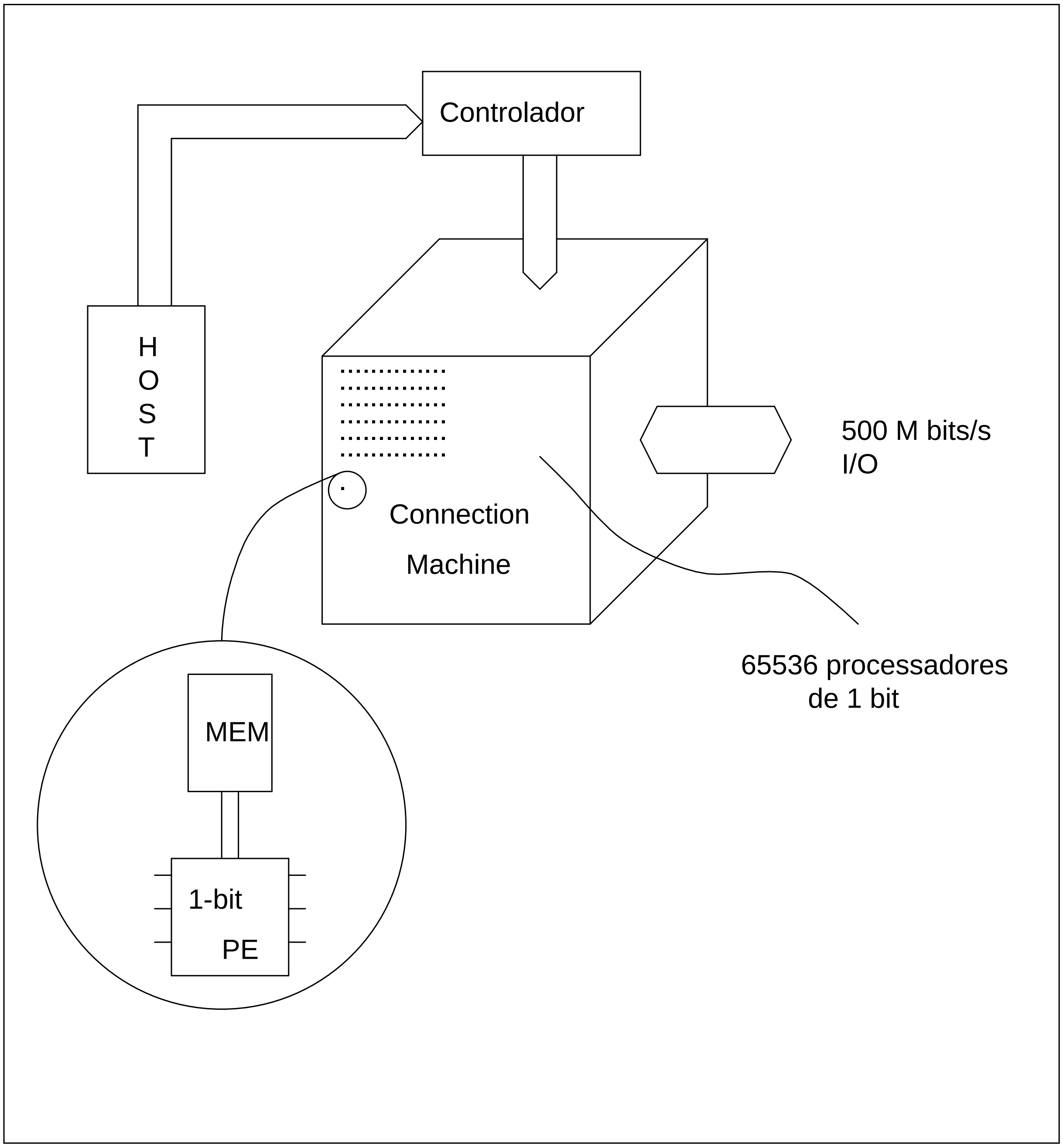

Figura 4.1: Arquitectura básica de uma Connection Machine

4.4.1 Partição de Dados

O desenvolvimento de um algoritmo baseado numa partição de dados de grão grosso, implica uma reestruturação dos dados do problema de forma a minimizar as suas necessidades em termos de comunicação. Para tal é necessária a criação de estruturas que agrupem os dados do problema em blocos de uma dimensão apreciável e exibindo uma forte interdependência.

A aproximação seguida no desenvolvimento deste algoritmo assenta na possibilidade de se poder encarar o circuito a simular como um aglomerado de subcircuitos, constituídos por um conjunto de nós fortemente ligados. O algoritmo consiste então na simulação de cada subcircuito individualmente, sempre que estejam reunidos um conjunto de condições que possibilite esse processamento.

O algoritmo de partição utilizado nesta versão é baseado numa estratégia de partição estática dos dados. No entanto, adiante se verá, que o algoritmo é capaz de, de forma automática, explorar a latência existente num circuito, assegurando assim uma divisão dinâmica do circuito em blocos activos e inactivos.

A ideia de base da partição efectuada, está intimamente relacionada com o controlo de simulação por acontecimentos que caracteriza o algoritmo e tenta extender o conceito de actividade localizada (vidé definição 3.1) ao caso de subcircuitos [Silveira87iii]. Uma vez mais, sugere-se que alterações numa dada área do circuito causarão em princípio apenas distúrbios locais, que demorarão um tempo finito a propagar-se para áreas mais distantes. Este conceito conduz a uma forma natural de resolução das equações e ao uso de técnicas de simulação controlada por acontecimentos [Neto87i]. A existência de múltiplos centros, exibindo uma actividade localizada, com fraca interacção, possibilita a exploração de paralelismo ao nível dos dados na simulação simultânea dos vários subcircuitos.

Se como resultado de uma partição dos dados de um circuito se obtiver um conjunto de subcircuitos, exibindo um mínimo de interacção, e constituídos por nós fortemente ligados, então variações internas a cada subcircuito podem ser resolvidas localmente e posteriormente propagadas à sua vizinhança. Desta forma minimizam-se as necessidades de comunicação entre subcircuitos, reduzindo-se estas apenas à propagação de perturbações entre grupos de nós fracamente ligados.

Motiva-se desta forma a introdução do conceito de **Fronteira de um Subcircuito**, de importância capital para a estratégia de partição a utilizar, bem como a de **Nó Fronteiro de um Subcircuito**.

Definição 4.2 (Fronteira de um Subcircuito)

A Fronteira de um Subcircuito é o conjunto de nós do subcircuito que exibem ligações com outros subcircuitos.

Definição 4.3 (Nó Fronteiro de um Subcircuito)

Um nó pertencendo à fronteira de um subcircuito é designado por Nó Fronteiro do subcircuito.

No algoritmo correntemente utilizado em **CINNAMON** (algoritmo 3.2), a solução de um acontecimento num nó implica a solução de um conjunto de equações associadas a esse nó e aos nós que lhe estão ligados. Desta forma, a solução de um acontecimento num nó fronteiro de um subcircuito implica o acesso a dados exteriores ao subcircuito. Introduz-se então a definição de **Faixa Fronteira de um Subcircuito** que por sua vez requer a definição de **Nó Quase-Fronteiro de um Subcircuito**.

Definição 4.4 (Faixa Fronteira de um Subcircuito)

A Faixa Fronteira de um Subcircuito é o conjunto de nós do subcircuito para os quais a resolução de um acontecimento implica o acesso a dados exteriores ao subcircuito. A Faixa Fronteira de um Subcircuito é constituída pelos Nós Fronteiros e pelos Nós Quase-Fronteiros desse subcircuito.

Definição 4.5 (Nó Quase-Fronteiro de um Subcircuito)

Um Nó Quase-Fronteiro de um subcircuito é um nó adjacente a um Nó Fronteiro, mas não pertencente à Fronteira do subcircuito.

Completando as definições anteriores, introduz-se o conceito de **Nó Interno de um Subcircuito** e de **Nó Externo do Circuito**[10].

Definição 4.6 (Nó Interno de um Subcircuito)

Um Nó Interno de um subcircuito é um nó do subcircuito que não pertence à Faixa Fronteira do subcircuito.

Definição 4.7 (Nó Externo de um Circuito)

Um Nó Externo de um Circuito é um nó do circuito ao qual se encontra ligado uma fonte de tensão independente.

Note-se que, face à definição anterior, os nós a que estão ligadas fontes de tensão não são considerados como pertencendo a nenhum subcircuito, sendo externos a todos eles. Acontecimentos em nós externos têm de facto um tratamento um pouco diverso, dado que podem causar a reavaliação de nós adjacentes, não sendo no entanto eles próprios recalculados, uma vez que a sua tensão é forçada pela fonte que se lhe encontra ligada. Num certo sentido, um nó externo é igualmente um nó fronteiro, sendo a diferença estabelecida por razões de simplicidade.

A partição de um circuito implica então a divisão dos seus nós em quatro conjuntos disjuntos: os fronteiros, os quase-fronteiros, os internos e os externos.

A introdução do conceito de subcircuito estende um pouco a hierarquia de dados do circuito. Da mesma forma que em **CINNAMON** cada nó tinha associado uma lista de nós adjacentes, é agora necessário manter associado a cada subcircuito uma lista dos subcircuitos vizinhos. Na realidade tal será realizado mantendo, para cada nó da fronteira de um subcircuito, uma lista dos subcircuitos a que esse nó está ligado, além da lista de nós de que já dispunha.

A existência de informação associada com os nós fronteiros, permitirá que perturbações na fronteira de um subcircuito sejam propagadas ao resto do circuito, de uma forma

[10]Note-se a diferença em que os nós Externos são considerados externos ao circuito global e logo a todos os subcircuitos, e os nós Internos são internos apenas a um subcircuito.

semelhante à propagação de perturbações nos nós em **CINNAMON**. Para cada nó é então necessária, além da informação a ele já associada, adicionar dados sobre o tipo de nó, o subcircuito a que pertence e, no caso de se tratar de um nó de fronteira, uma lista dos subcircuitos a que se encontra ligado.

A questão fundamental que ainda não foi abordada prende-se com a forma como a partição de um circuito se deve processar, ou equivalentemente, como definir o que deve ser um subcircuito. De forma geral, um subcircuito deve ser uma colecção de nós, exibindo ligações fortes entre si e fracamente ligados ao exterior. Uma estratégia típica de partição de circuitos, vulgarmente utilizada em simuladores ao nível de interruptor [Bryant80] baseia-se na definição seguinte:

Definição 4.8 (Subcircuito ou Grupo)

Um Subcircuito de um circuito MOS é constituído por um conjunto de nós ligados por resistências, condensadores ou canais de transistores. Um subcircuito satisfazendo estas condições é vulgarmente designado por Grupo.

Esta estratégia conduz tipicamente a grupos de pequenas dimensões[11], o que não satisfaz os requisitos pretendidos, uma vez que nesse caso cada subcircuito terá um número muito reduzido de nós internos e as necessidades de comunicação com outros subcircuitos serão demasiado elevadas. Um exemplo duma partição deste tipo é apresentado na figura 4.2.

Uma algoritmo de partição baseado nesta estratégia é utilizado em **CINNAMON**, no decorrer da análise *DC*, para acelerar a convergência da mesma[12]. Um algoritmo de partição do circuito mais adaptado a uma versão paralela de grão grosso foi desenvolvido a partir da estratégia já utilizada. Assim, começa-se por se proceder a uma partição em grupos de acordo com a definição 4.8, efectuando-se em seguida um reagrupamento dos subcircuitos obtidos de forma a aumentar as dimensões de cada grupo e simultaneamente diminuir o número total de grupos. Isto permite uma certa versatilidade na definição de subcircuito, mantendo compatibilidade com a versão sequencial.

[11]Para circuitos típicos os grupos derivados duma estratégia deste tipo têm, em geral, na ordem de 2 a 4 nós.

[12]Tal não foi abordado na secção 3.3.2, em que apenas se aludiu ao algoritmo básico do simulador e não aos detalhes de implementação.

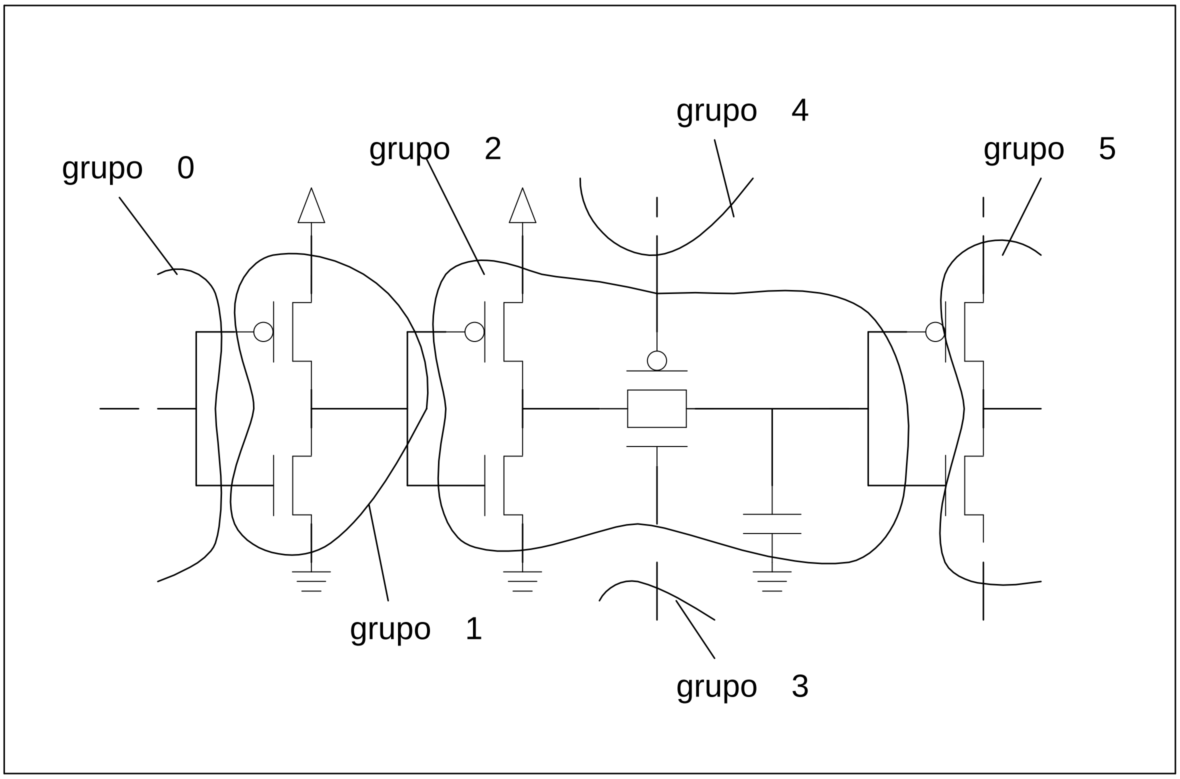

Figura 4.2: Divisão de um circuito em grupos de acordo com a definição 4.8.

O algoritmo que reagrupa os subcircuitos é um algoritmo do tipo "guloso", que encara cada subcircuito como um nó de um grafo não dirigido e prossegue coleccionando grupos, mantendo um equilíbrio entre o número de nós em cada subcircuito assim criado. No final, devolve um conjunto de subgrafos, os subcircuitos, em número controlável pelo utilizador face ao número de unidades computacionais disponíveis[13]. Este algoritmo é apresentado de seguida (algoritmo 4.4).

[13]A existência de uma relação entre o número de processadores disponíveis e o número de subcircuitos pretendidos não é na verdade um assunto completamente esclarecido, sendo actualmente alvo de mais aturado estudo.

Algoritmo 4.4 (Partição de um Circuito em Grupos)

```
partição()              /* para uma versão de grão grosso */
{
    partição inicial do circuito em grupos de acordo com a definição 4.8;
    reagrupamento de subcircuitos;
    reajustamento dos dados relativos à adjacência entre grupos;
}
```

Idealmente, o reagrupamento das partições inicialmente obtidas em subcircuitos de maiores dimensões, deve ser efectuado de forma a minimizar a comunicação entre subcircuitos. Não é no entanto actualmente conhecido nenhum algoritmo que endereçe esse problema de forma concreta, pelo que o tópico constitui assunto de pesquisa a desenvolver futuramente. O mesmo se aplica em relação ao número óptimo de subcircuitos a obter após reagrupamento, o qual está no entanto de alguma forma relacionado com o número de processadores disponíveis.

Após a partição do circuito introduziu-se assim um novo nível na hierarquia de dados, constituído pelos subcircuitos como entidades próprias, os quais, na medida do possível, poderão ser simulados em simultâneo de forma a permitir o aumento de desempenho da simulação.

4.4.2 Conceitos Algorítmicos

A partição de um circuito possibilita assim a sua simulação concorrente através da existência de diversos processadores em competição para acesso aos dados de cada subcircuito. É necessária então a existência de um mecanismo que garanta o acesso exclusivo a esses mesmos dados. Acresce que o controlo por acontecimentos implica que os vários processadores podem estar a simular em instantes diferentes e a consistência dos dados tem de ser garantida de forma a que os processadores mais lentos não vejam esses dados corrompidos por processadores mais rápidos.

Note-se que os simuladores de circuitos baseados na discretização do eixo dos tempos, por construção evitam este problema. De facto, mesmo que se proceda a uma partição do circuito e cada processador se ocupe de um conjunto de dados (no caso um subcircuito), todos eles funcionarão sincronamente e no mesmo tempo de simulação. Na pior das hipóteses, dependendo do algoritmo, poderá ser necessário iterar os resultados até ser atingida a convergência das tensões em cada instante de tempo, mas a consistência dos dados do circuito é garantida entre cada dois pontos no eixo dos tempos. No caso da versão baseada em discretização temporal, descrita na secção 4.3, esse processo é ainda mais simplificado, sendo desnecessário iterar em cada instante, uma vez que se utiliza um algoritmo explícito.

Apesar deste problema, muitos algoritmos baseados em controlo por acontecimentos têm sido implementados sobre máquinas paralelas. A maioria, no entanto, são simuladores ao nível lógico, adicionando assim alguma simplificação ao problema uma vez que o modelo a eles inerente é um modelo unidireccional, e exploram paralelismo em cada instante de tempo, logo de forma síncrona[14].

Ao considerar as opções possíveis para controlo do processo de simulação, torna-se claro que as técnicas usuais, utilizadas por exemplo nos simuladores lógicos, não são aplicáveis neste caso. Na maioria das implementações de algoritmos de simulação conduzida por acontecimentos, é usual impôr uma restrição denominada por **Restrição de Precedência**. Tal relação implica que cada grupo ou partição apenas pode ser avaliado no instante $t + \Delta t$ se todas as suas entradas forem conhecidas no instante t. Esta restrição não tem equivalente num algoritmo de simulação ao nível eléctrico uma vez que não é possível impôr unidireccionalidade na interacção entre os elementos do circuito, o que torna impossível a definição de entradas de um subcircuito. Por outro lado, basear a decisão sobre a resolução de um dado acontecimento apenas no tempos de transição dos nós a ele ligados traz efeitos curiosos, uma vez que nesse caso se obteria um desempenho igual ao de uma versão sequencial[15].

Para solucionar este problema é conveniente considerá-lo sob um ponto de vista diferente. Se se suposer que o valor de todos os nós é conhecido em todos os instantes, não existe nenhuma restrição de precedência a impôr e cada partição pode ser simulada independentemente de todas as outras [Arnold85]. Neste tipo de **Simulação Caótica** [Chazan69], cada partição é simulada concorrentemente, assumindo que os valores de que necessitará são conhecidos em qualquer instante. Sempre que houver uma alteração na fronteira de uma partição, todas as partições que porventura possam ter que usar esse valor, terão que ser avisadas, e possivelmente refazer os seus cálculos, caso descubram que o fizeram com valores errados.

[14]Desta forma, os simuladores lógicos baseados em controlo por acontecimentos apresentam características muito semelhantes às da implementação por discretização temporal anteriormente descrita.

[15]De facto impondo-se uma tal restrição, apenas um nó se poderia mover de cada vez, obtendo-se na verdade uma versão sequencial à qual se associou o *overhead* de mais um nível hierárquico.

Para possibilitar esta operação, cada subcircuito tem que, de alguma forma, memorizar os dados usados e para tal, necessita de implementar uma **Estratégia de Memorização** *(Checkpoint Strategy)* de forma a que possa guardar o seu estado durante certos intervalos de tempo. Um **Procedimento de Retrocesso** *(Rollback)*, permitirá recuar no tempo e recuperar, com base na lista de *checkpoints*, o estado correcto antes do cálculo incorrecto ter sido efectuado e continuar a simulação. Esta estratégia foi implementada na versão corrente, sendo para cada um dos grupos, mantida uma lista de *checkpoints*, com todas as alterações no estado dos seus nós. Foram igualmente desenvolvidos mecanismos para efectuar o *rollback* de cada subcircuito no tempo. Estes serão abordados adiante.

Uma diferença importante deve ser no entanto realçada. Nos grupos resultantes da partição, todos os nós fronteira são simultaneamente entradas e saídas. Se um nó fronteira de um grupo sofre uma alteração, então todos os seus vizinhos têm que ser avisados e terão, provavelmente, que fazer *rollback* do seu estado. Cada subcircuito vizinho do anterior, após fazer *rollback* da sua simulação, irá refazer os cálculos do seu lado da fronteira e, se houver uma mudança, irá também avisar os seus vizinhos, incluindo o grupo que primeiramente o avisou. Esta sequência pode tornar-se desagradável, particularmente no caso do primeiro grupo ter recomeçado a sua actividade, ou no caso de os nós fronteira de ambos os lados sofrerem grandes alterações e o passo de tensão ser reduzido.

Afim de evitar tanto quanto possível esta situação impôs-se uma restrição que se designará por **"Restrição na Fronteira"** e cuja definição se segue.

Definição 4.9 (Restrição na Fronteira)

Se, dentro de uma partição, ocorrer um acontecimento num nó fronteira, esse acontecimento será resolvido se e só se a partição for a detentora do tempo global.

Definição 4.10 (Tempo Global)

Tempo Global é, em cada instante, o mínimo dos tempos correntes de simulação dos vários subcircuitos.

Com esta restrição, quando um acontecimento num nó fronteira está em vias de ser calculado, o processador que nesse instante está encarregado desse subcircuito, tem que se assegurar que todos os acontecimentos nos outros subcircuitos (quer estejam a ser simulados ou marcados na lista de acontecimentos) e o próximo acontecimento de fonte de tensão, são posteriores no tempo. Se tal for o caso, é possível resolver o acontecimento; caso contrário, é necessário parar a simulação desse grupo, colocá-lo na lista no instante correcto e iniciar o processamento de outro grupo[16].

A estratégia apresentada tem consequências úteis e importantes além das já referidas. O facto de apenas o subcircuito mais lento ser autorizado a resolver acontecimentos na fronteira, leva a que, se um subcircuito descobre que é o mais lento no instante t, então também sabe imediatamente que todo o seu "passado" está correcto, assim como estão correctos os cálculos efectuados até ao instante t para todos os outros subcircuitos[17].

O processo descrito, consiste de facto numa forma implícita de sincronização entre grupos, (**Auto-sincronização ou Sincronização Local**), revelando-se no entanto bastante eficiente. Note-se que essa sincronização se processa de forma automática, não requerendo, como é usual, que todos os processadores parem sempre que se leva a cabo uma sincronização geral.

Desta forma, cada grupo continua internamente a sua simulação, sem se preocupar com os outros grupos, desde que os acontecimentos que resolve sejam respeitantes a nós internos. Quando um acontecimento fronteira é resolvido, os seus vizinhos têm que tomar conhecimento dessa alteração. Note-se que ao resolver um acontecimento fronteira se manipulam e alteram dados que não são necessariamente do subcircuito (dados partilhados) e para assegurar a consistência desses mesmos dados, apenas um processador pode fazê-lo de cada vez.

Isto introduz um outro conceito relacionado com a necessidade de dar conhecimento aos grupos vizinhos de qualquer alteração na fronteira comum. Esses subcircuitos devem então verificar se não fizeram cálculos baseados em dados incorrectos. Introduz-se então o conceito de **Ordem** para caracterizar este mecanismo de passagem de mensagens.

[16]Repare-se que neste caso existe uma probabilidade elevada que o primeiro subcircuito presente na lista de acontecimentos seja o mais atrasado no tempo.

[17]Uma vez que se trata do subcircuito mais lento, isso impossibilita que qualquer outro subcircuito possa estar num instante anterior a t, portanto todos estão correctos até t .

Definição 4.11 (Ordem)

Uma Ordem é uma mensagem enviada de um subcircuito que resolveu um acontecimento num nó Fronteira, para todos os subcircuitos que partilham essa fronteira. Estes poderão ter que refazer os seus cálculos face a essa alteração de estado.

A escolha da designação deriva directamente da acção efectuada pelo emissor da mensagem que, no fundo, ordena ao receptor que ele deve refazer um cálculo. O emissor logicamente não terá que recuar no tempo, enquanto que o receptor possivelmente terá que o fazer. Note-se que se um dado grupo descobre, no instante t, que é o mais lento, envia uma ordem contendo o instante t e o nó que se alterou. O facto da ordem incluir o instante, implica que cada nó apenas necessita ter espaço para uma ordem,. Isto tem como consequência que após receber uma ordem no instante t, antes da resolução dessa ordem, nenhum outro grupo pode enviar outra ordem num instante maior que t, pois isso seria uma violação da **Restrição na Fronteira**. É portanto necessário cuidado ao determinar o tempo de simulação de um subcircuito, dado que se deve verificar se existe alguma ordem pendente.

Impõe-se uma explicação sobre o significado de uma ordem: quando se envia uma ordem após resolver um acontecimento fronteira, faz-se uma nova marcação desse nó e dos seus adjacentes, mas apenas daqueles que pertencem ao grupo em questão, informando-se os restantes grupos de que devem efectuar esse cálculo. Estes, ao receber a ordem, completam a resolução daquele acontecimento, recalculando os nós adjacentes ao nó activo, que pertencem a esse grupo.

Ao receber uma ordem, um subcircuito sincroniza-se imediatamente com o processador/grupo mais lento e sabe imediatamente que todos os seus cálculos estão correctos até ao instante da ordem. Isto pode ser utilizado para arquivar os resultados das simulações efectuadas até esse instante.

Uma vez introduzidos os conceitos básicos, a secção seguinte descreverá as restantes opções de projecto e o algoritmo de controlo implementado.

4.4.3 Algoritmo de Controlo

O algoritmo de controlo da simulação utilizado nesta versão estende os conceitos introduzidos em **CINNAMON** (algoritmo 3.2), pela introdução de mais um nível hierárquico.

Trata-se ainda de um algoritmo baseado em técnicas de controlo por acontecimentos, mas em que a lista global de nós, é substituída por várias listas parciais, uma para cada subcircuito. A um nível mais elevado, existe uma lista de subcircuitos, ordenada de forma crescente pelo tempo de simulação de cada subcircuito.

Sempre que se liberta um processador, ele procura o primeiro grupo dessa lista, e inicia a sua simulação. A estratégia para *scheduling* dos processadores aos grupos é portanto aleatória, o que se torna apropriado numa máquina com esta arquitectura e simultaneamente bastante rápido. Quando um processador liberta um grupo, insere-o na lista, no local apropriado, se o grupo ainda tiver acontecimentos marcados no futuro ou tiver recebido uma ordem, caso contrário simplesmente larga-o, deixando-o *não agendado*. Desta forma procede-se a uma partição dinâmica do circuito, uma vez que apenas os subcircuitos que estão na lista, são candidatos a ser simulados, ou seja, apenas aqueles nos quais há actividade no momento. O acesso à lista é feito através de um semáforo, ou *lock*, que assegura que os dados globais não são acedidos para escrita ao mesmo tempo por mais do que um processador, nem alterados quando alguém os possa estar a ler.

Além de uma lista de nós vizinhos associados a cada nó fronteira, é também necessário dispôr de alguma informação volátil associada a cada grupo: o número do nó, uma indicação do estado do grupo ("em simulação", "marcado" ou "desmarcado"), o número do processador[18] que o simula (quando em simulação), uma lista de acontecimentos para os nós pertencentes ao gru po, uma estrutura "ordem" funcionando como um endereço onde os grupos vizinhos podem deixar ordens e uma lista de *checkpoints* que permita, se necessário, recuperar o estado da simulação num instante passado. Cada uma destas duas últimas estruturas, dado que se trata de dados partilháveis entre processadores, tem associado um semáforo, que garante a exclusividade no seu acesso. Todas estas estruturas são portanto criadas ao nível dos subcircuitos, pelo que se mantém uma compatibilidade total de dados com **CINNAMON** em níveis inferiores da hierarquia.

Os "semáforos internos" são a forma utilizada para sincronizar o acesso aos dados globais, sendo alterados através das chamadas ao sistema apresentadas na secção 3.4. Existe ainda uma outra estrutura que deve ser mantida para além da lista de acontecimentos de cada grupo. Está relacionada com o estado dos processadores, e deve indicar o seu estado ("activo" ou "inactivo") e conter a informação (o número) do grupo que está a simular, quando em actividade. Estes dados são úteis quer quando se pretende determinar o grupo mais lento, quer para determinar quando se deve terminar a concorrência.

Para a implementação descrita, teve basicamente que se reescrever 80% do código do simulador (não considerando o código relacionado com a entrada do "parser" e as

[18] Mais correctamente neste caso deveria dizer-se o número da tarefa na sequência de tarefas criadas.

rotinas de cálculo dos modelos dos dispositivos, que não estão directamente ligadas ao controlo da simulação). As rotinas que fazem a gestão dos grupos e das listas foram completamente reescritas, as que lidam com acontecimentos nos nós apenas modificadas. As rotinas básicas do algoritmo de controlo da simulação são novas, desde a marcação dos processadores até à marcação de acontecimentos ou ordens e resolução dos mesmos.

Segue-se uma descrição muito geral do algoritmo implementado[19]. Na secção seguinte descreve-se a estratégia implementada para *checkpoint*, e como essa informação é utilizada para recuperar o estado de um nó ou de um grupo num dado instante.

[19] O apêndice E consiste numa versão um pouco mais detalhada do algoritmo.

Algoritmo 4.5 (Cinnamon: Versão Paralela de Partição de Dados)

```
/* Admitindo uma configuração com N processadores */
simulação()              /* seja DC ou Transiente */
{
    t = 0 ;
    NúmeroGrupos = partição_do_circuito(NúmeroÓptimo);
    marcação das fontes de tensão;
    concurrent_call(..., simula_grupos, N);
}

simula_grupos(processador)
{
    enquanto( há grupos para simular e t < TempoMáximo) {
        retira o primeiro grupo da lista;
        enquanto( há acontecimentos na lista e t < TempoMáximo) {
            lê a informação do próximo acontecimento(nó, tempo, tensão)
              ;
            t = tempo ;
            se o acontecimento é num nó na Faixa Fronteira {
                se o acontecimento não pode ser simulado, sai;
                caso contrário actualiza o arquivo local;
            }
            lineariza os dispositivos ligados ao nó activo;
            determina a próxima alteração(nó, lista de nós adjacentes);
            guarda informação do acontecimento localmente(nó, tempo,
              tensão);
        }
    }
}
```

4.4.4 Manutenção de Consistência dos Dados

A lista de *checkpoints* contém toda a informação necessária para permitir retroceder no tempo e recuperar o estado de um subcircuito num dado instante. Deve portanto conter toda a informação relevante sobre o processo de simulação.

Nesta implementação, além do conceito de acontecimento, existe igualmente o conceito de "ordem", como um meio de controlo da simulação. No entanto, a imposição da Restrição na Fronteira definida anteriormente, torna desnecessária a manutenção de informação relativa a ordens de outros subcircuitos, sendo estas utilizadas para actualizar a lista de *checkpoints*, dado que, como vimos, servem como mecanismos de sincronização.

Desta forma, apenas se guarda nessa lista informação referente a alterações na lista de acontecimentos de cada grupo, em vez de se guardar uma descrição completa do estado da mesma em cada instante. Desta forma a lista de *checkpoints* é alterada sempre que se insere, ou retira um acontecimento da lista de acontecimentos Esta metodologia é de facto mais apropriada para um algoritmo deste tipo em que as alterações nos dados não são efectuadas periodicamente, nem seguem nenhum padrão bem definido, ao contrário de um algoritmo baseado numa discretização do eixo dos tempos [Arnold85].

Se em qualquer instante, como resultado de uma ordem, se pretender recuar no tempo de simulação e recuperar o estado dum subcircuito num instante no passado, o procedimento é extremamente simples, bastando percorrer a lista de *checkpoints* até esse instante[20] e repôr o conteúdo da lista de acontecimentos refazendo, em sentido inverso, todas as alterações indicadas na lista de *checkpoints*.

O processo de activação do mecanismo descrito anteriormente é igualmente bastante simples. Note-se que até receber uma ordem, cada subcircuito simula os seus nós internos independentemente dos outros subcircuitos, pelo que o processo de *rollback* apenas pode ser desencadeado pela chegada de uma ordem. Após a recepção de uma ordem, contendo informação de alteração num nó da fronteira comum aos subcircuitos emissor e receptor, este deve verificar se essa alteração pode comprometer os resultados de cálculos efectuados anteriormente, e apenas nesse caso deve iniciar o processo de *rollback*.

---

[20]Na verdade, dado que se trata de tempo contínuo, procura-se a primeira alteração posterior ao instante a que a ordem se refere.

Capítulo 5

Exemplos e Análise de Desempenho

Neste capítulo apresentam-se alguns resultados de simulações efectuadas com os protótipos descritos, sobre um conjunto de circuitos previamente seleccionados. De início descrevem-se muito genericamente os testes efectuados, adiantando-se razões para a escolha dos vários circuitos, apresentando-se em seguida os resultados obtidos.

Os tempos de simulação apresentados foram medidos directamente sobre os programas, utilizando um conjunto de subrotinas existentes no sistema UNIX. Os resultados indicados referem-se a tempos de simulação total e tempos medidos durante a análise transitória. Atendendo a que os primeiros incluem o tempo relativo ao *parsing* e a tarefas de inicialização, os últimos são mais relevantes para a análise dos algoritmos implementados.

5.1 Circuitos de Teste

O conjunto de circuitos de teste utilizados *(benchmarks)* foi escolhido de entre a biblioteca de circuitos existentes no grupo de Simulação do INESC. Note-se que, tratando-se de implementações numa máquina paralela, se pretendem circuitos de uma dimensão apreciável, suficiente para permitirem uma aceleração visível. Desta forma foram escolhidos os seguintes circuitos, cuja dissimilaridade aumenta a nossa perspectiva sobre as capacidades dos protótipos:

- "100inv": uma cadeia de 100 inversores CMOS.
- "inv3par": três cadeias paralelas, cada uma com 100 inversores.

- "inv6par": seis cadeias paralelas, cada uma com 100 inversores.
- "a16": uma unidade aritmetico-lógica de 16 bits desenhada segundo a metodologia proposta por Mead & Conway [Mead80].
- "a32": uma unidade aritmetico-lógica de 32 bits desenhada segundo a metodologia proposta por Mead & Conway [Mead80].
- "compt": comparador lógico de 4 bits.
- "mem4": uma memória dinâmica de 4 bits.
- "ring": um oscilador em anel de 19 estágios.

Os testes efectuados sobre os circuitos anteriores foram os seguintes:

- Cadeias de Inversores: propagação de um sinal ao longo da cadeia provocando alteração lógica nas saídas de todos os inversores; serão designados pelo nome do circuito.
- Unidades Aritmético-Lógicas:
  - Teste 1 - pior caso de propagação de *transporte* da soma ao longo de toda a cadeia; será designado pelo nome do circuito.
  - Teste 2 - adição de 2 números de 32 bits na ALU de comprimento correspondente; sem geração de *transporte*; será designado por al32.
- comparador: comparação entre dois números lógicos numa situação aleatória.
- memória: leitura e escrita de 1 bit em cada elemento da memória.
- oscilador: oscilação da cadeia durante 3 ciclos completos.

De particular importância é o conhecimento da dimensão de cada circuito, tanto em termos de nós como de dispositivos. Esses dados são apresentados na tabela 5.1.

Os circuitos mencionados e os testes propostos não apresentam necessariamente um elevado grau de paralelismo de dados ou funcional claro. No capítulo seguinte, os resultados aqui obtidos serão analisados e cada teste efectuado será debatido de forma mais profunda, para que se possa entender mais claramente o significado dos resultados aqui apresentados.

Circuito	N^0 de nós	N^0 de Transistores
100inv	103	200
inv3par	303	600
inv6par	603	1200
a16	594	996
a32	1170	1988
compt	122	224
mem4	31	50
ring1	22	39

Tabela 5.1: Dimensões dos circuitos de teste

Teste	Cinnamon (seg.)	Vers. Acelerada (seg.)	η
100inv	19	11	1.73
inv3par	59	34	1.74
inv6par	123	75	1.64
a16	146	82	1.78
a32	164	102	1.61
al32	52	48	1.08
compt	234	135	1.73
mem4	10	7	1.43
ring	14	9	1.56

Tabela 5.2: Versão Acelerada: Tempo Total de Simulação

5.2 Resultados: Versão Acelerada

Para testar a versão acelerada foram conduzidos os testes indicados anteriormente, cujos resultados estão sumarizados nas tabelas 5.2 e 5.3. A tabela 5.2 refere-se ao tempo total de simulação, enquanto que a tabela 5.3 se refere aos tempos da análise transitória. Nestas tabelas é igualmente apresentado o tempo de simulação obtido com **CINNAMON**, sendo apresentado o factor de aceleração, η, definido como o quociente entre os tempos de simulação paralela e sequencial (ver definição 4.1). De facto, tratando-se de uma versão acelerada do algoritmo, em que o paralelismo foi explicitamente explorado sem alteração da sequência de execução do simulador e atendendo a que a arquitectura do *Alliant* se enquadra nas características consideradas como ideais para este protótipo,

Teste	Cinnamon (seg.)	Vers. Acelerada (seg.)	η
100inv	18	9	2.00
inv3par	53	29	1.83
inv6par	111	62	1.79
a16	126	64	1.97
a32	121	59	2.05
al32	9	6	1.50
compt	232	132	1.76
mem4	9	6	1.50
ring	13	8	1.63

Tabela 5.3: Versão Acelerada: Tempo de Simulação de Transiente

a razão de aceleração, calculada da forma indicada, pode ser considerada como a forma mais correcta de medir o seu desempenho.

5.3 Resultados: Versão de Discretização Temporal

Teste	efit(seq.) (seg.)	efit(par.) (seg.)	η
100inv	248	97	2.56
inv3par	760	311	2.44
inv6par	1482	672	2.21
a16	962	377	2.56
a32	3304	1231	2.68
al32	266	114	2.33
compt	1199	441	2.72
mem4	49	38	1.23
ring	55	26	2.12

Tabela 5.4: Discretização Temporal: Tempo Total de Simulação

Para testar a versão protótipo baseada em discretização temporal foram conduzidos os testes indicados anteriormente, cujos resultados estão sumarizados nas tabelas 5.4 e 5.5. De novo, a tabela 5.4 refere-se ao tempo total de simulação, enquanto que a tabela 5.5 se refere aos tempos da análise transitória. Nestas tabelas é igualmente apresentado o factor

Teste	efit(seq.) (seg.)	efit(par.) (seg.)	η
100inv	237	92	2.57
inv3par	725	297	2.44
inv6par	1416	640	2.21
a16	859	330	2.60
a32	3109	1144	2.72
al32	74	27	2.74
compt	1189	437	2.72
mem4	48	37	1.30
ring	55	25	2.20

Tabela 5.5: Discretização Temporal: Tempo de Simulação de Transiente

de aceleração, η, que neste caso se refere à razão entre os tempos de simulação das versões sequencial e paralela baseadas em discretização temporal. Tratando-se neste caso de uma versão baseada num algoritmo apresentando diferenças importantes relativamente ao algoritmo de **CINNAMON** seria incorrecto, no nosso ponto de vista, definir a eficiência desta implementação relativamente à versão original do simulador. De facto, o algoritmo desenvolvido apenas tem em comum com **CINNAMON** a utilização de técnicas explícitas de aproximação por exponencial, diferindo radicalmente deste em termos de controlo de simulação. Acresce que a arquitectura do *Alliant*, embora oferecendo uma plataforma de teste das potencialidades do algoritmo, afasta-se consideravelmente da arquitectura considerada como mais indicada para este tipo de algoritmos (vidé secção 4.3.3). Desta forma, a razão de aceleração apresentada foi definida como relativamente à versão sequencial de discretização temporal, cujos resultados são igualmente apresentados sob a designação **efit**.

5.4 Resultados: Partição de Dados

Os resultados que se seguem foram obtidos através de um protótipo baseado em paralelismo de dados de grão grosso, como descrito na secção 4.4. Esta versão encontra-se ainda numa fase preliminar do seu desenvolvimento, devendo os resultados apresentados ser considerados como meros indicadores do tipo de aproximação explorada. Os resultados apresentados na tabela 5.6 referem-se a tempos de simulação obtidos com o protótipo existente, sendo o seu desempenho medido relativamente à sua execução supondo um número crescente de processadores. O factor de aceleração apresentado é assim definido,

Teste	pcin(1 proc) (seg.)	pcin(3 procs.) (seg.)	η
100inv	26	17	1.53
inv3par	81	29	2.79
inv6par	166	60	2.77
a32	161	126	1.28
al32	264	136	1.94
compt	499	346	1.44
mem4	10.9	6.7	1.63

Tabela 5.6: Partição de Dados: Tempo de Simulação Transitória

relativamente ao desempenho do mesmo protótipo, mas num subconjunto da máquina constituído por apenas um processador. Atendendo ao estado de desenvolvimento desta versão parece-nos ser essa a forma mais correcta de medir a eficiência de paralelização que se conseguirá com este algoritmo.

5.5 Crítica e Análise de Resultados

Os resultados apresentados nas secções anteriores são extremamente encorajadores quanto às possibilidades dos algoritmos desenvolvidos.

Atendendo a que os circuitos de teste são circuitos de grandes dimensões, é conveniente utilizar os resultados da análise transitória como medida do desempenho de cada versão, dado que no tempo total de simulação está incluído o tempo de leitura dos dados do circuito e da análise DC efectuada anteriormente à análise transitória.

No que respeita à versão acelerada e reportando-nos aos tempo de simulação da análise transitória, podemos ver que a aceleração média conseguida foi próxima de dois, o que representa um valor muito razoável, principalmente atendendo às limitações desta versão. De facto, recorde-se que nesta versão apenas foi explorado paralelismo ao nível da linearização dos dispositivos e na resolução de cada acontecimento, pelo que as restantes tarefas, incluindo a gestão da lista de acontecimentos e a interacção com as fontes independentes do circuito, são efectuadas de forma sequencial.

Em particular, na linearização de dispositivos, apenas é possível linearizar três dispositivos em simultâneo (atendendo à configuração do *Alliant* utilizado), pelo que se o número de dispositivos ligados a cada nó não é um múltiplo de três, haverá um desperdício de recursos computacionais, dado que um ou mais processadores estarão inactivos enquanto o restante executa o código relativo a essa linearização. Note-se que isto se repete sempre que esse nó sofrer uma perturbação, pelo que este desiquilíbrio de carga pode ser substancial. O mesmo problema surge quando se recalcularem os nós adjacentes a um nó activo, pelo que a conjugação dos dois problemas, leva a que se deva considerar uma aceleração de dois como um resultado bastante aceitável.

É interessante notar ainda, que a aceleração conseguida com esta versão não é proporcional à dimensão dos circuitos, sendo na verdade ditada pelos factores mencionados acima e pela conectividade média de um nó em termos de dispositivos e nós.

A versão acelerada proporciona assim uma aceleração real do processo de simulação, sendo de grande interesse para uma máquina com as características arquitectónicas do *Alliant* e com um número reduzido de processadores. Mesmo em termos de tempo total de simulação pode-se observar que a aceleração conseguida é de molde a tornar apreciável a redução no tempo de simulação. A grande desvantagem desta versão, no entanto, é que se se dispuser de uma máquina com um número elevado de processadores, não é possível utilizar esses recursos de forma cabal, dado que o paralelismo explorado não permite a utilização de um número de processadores em média superior a 4 ou 6.

Para tal, como descrito, foi desenvolvida a versão baseada em paralelismo temporal, cujos resultados são ainda mais encorajadores. De facto a aceleração conseguida em quase todos os circuitos é superior a 2.5, o que excede as previsões iniciais, essencialmente atendendo a que, nesta versão, o paralelismo é explorado ao nível da estrutura elementar, o nó, e que todas as acções não directamente relacionadas com a actividade de um nó são executadas sequencialmente. Estão neste caso a escrita dos resultados da simulação, a interpolação do valor das fontes para cada instante de tempo e a gestão da pseudo-lista de actividade.

De facto, um cálculo elementar, baseado na assumpção de que a percentagem do tempo relativo à escrita dos resultados da simulação não excede 25% do tempo total de simulação e admitindo que o restante código foi paralelizado de forma perfeita, não se esperaria mais do que uma aceleração de 2 relativamente à versão sequencial, dado que, neste caso:

$$\eta = \frac{0.75 \; Tempo \, Total + 0.25 \; Tempo \, Total}{0.75/3 \; Tempo \, Total + 0.25 \; Tempo \, Total} = 2 \tag{5.1}$$

Os resultados obtidos excedem este valor, o que se explica esclarecendo que a versão utilizada como base para o desenvolvimento do algoritmo de discretização temporal engloba uma estratégia destinada a reduzir o tempo necessário para escrita dos resultados da simulação, e que proporciona uma redução do mesmo para valores inferiores a 10% do tempo total de simulação. Mesmo nesta situação e considerando inclusivé o tempo total de simulação a expectativa é grande quanto às potencialidades de um tal tipo de algoritmo, numa arquitectura com um muito maior número de processadores, uma vez que a aceleração possível é apenas limitada pela dimensão do circuito, ou seja pelo número de nós e dispositivos do mesmo. É igualmente de realçar a simplicidade dos conceitos envolvidos neste tipo de aproximação, bem como na sua implementação, uma vez que o paralelismo é explorado ao nível das estruturas físicas elementares, os nós do circuito.

Os resultados apresentados demonstram claramente a validade de um tal tipo de aproximação podendo ver-se que exceptuando os circuitos com pouca actividade simultânea, como o caso da unidade de memória, a aceleração conseguida é básicamente a máxima possível. Será também interessante, testar esta versão numa máquina com uma arquitectura verdadeiramente distribuída, de forma a determinar de que forma a aceleração conseguida será diminuída pelo *overhead* relativo à comunicação entre processadores numa máquina desse tipo. Tal não foi efectuado até ao momento devido à impossibilidade de conseguir acesso a uma máquina com uma arquitectura desse tipo.

Relativamente à versão de partição de dados e atendendo ao seu estado de desenvolvimento optou-se por estimar a validade das opções tomadas no seu desenvolvimento comparando os tempos obtidos pela versão paralela executando em subconjuntos da máquina com número variável de processadores. Os resultados obtidos parecem indicar que é necessário um estudo mais profundo das diversas opções tomadas neste tipo de aproximação, embora os resultados disponíveis para os circuitos de maiores dimensões pareçam sugerir a existência de um elevado grau de paralelismo. Note-se que de acordo com a aproximação seguida no estabelecimento desta versão ela se aplica melhor a circuitos em que é fraca a interacção entre blocos, de forma a minimizar a comunicação entre os subcircuitos resultantes da partição.

A elevada aceleração conseguida para alguns circuitos leva a considerar como favorável o tipo de aproximação desenvolvido, sendo no entanto necessário um estudo mais aturado dos algoritmos de partição, responsáveis pelo menor desempenho do protótipo relativamente a outros circuitos. Em particular nos circuitos que apresentam elevada actividade e em que esta é bastante localizada, existindo fraca ligação entre os diversos blocos do circuto, como o caso das cadeias de inversores, ou o da unidade aritmético-lógica, no caso da adição sem geração de *transporte*, essa aceleração é bastante elevada. No caso geral, o factor de paralelização obtido é portanto suficiente para aconselhar um desenvolvimento mais profundo deste tipo de aproximação.

Em geral, e como conclusão, as versões desenvolvidas parecem fornecer indicações encorajadoras, tendo no nosso entender sido demonstrada a possibilidade de obtenção de uma aceleração real do processo de simulação eléctrica de circuitos através da exploração de paralelismo ao nível do algoritmo.

Capítulo 6

Considerações Finais

Neste capítulo conclui-se a descrição do trabalho realizado sumarizando os resultados obtidos e apontando um conjunto de possíveis desenvolvimentos a realizar como continuação do projecto agora iniciado.

6.1 Sumário do Trabalho Realizado

Este trabalho focou o estudo e desenvolvimento de algoritmos de simulação eléctrica de circuitos para implementação em arquitecturas paralelas de computadores.

Os simuladores desenvolvidos no decurso deste trabalho destinam-se a ser integrados num pacote de ferramentas de apoio ao projecto, servindo simultaneamente como base de estudo para futuras implementações paralelas de outro tipo de algoritmos de CAD.

No primeiro capítulo introduziu-se o problema em estudo motivando-se a necessidade de simulação ao nível eléctrico e de utilização de meios computacionais mais eficientes. Nesta ordem de ideias introduziu-se o conceito de processamento paralelo como ambiente de desenvolvimento de algoritmos de CAD de elevado desempenho.

O segundo capítulo versou o tema de arquitecturas de processamento paralelo tendo o mesmo sido debatido ao nível arquitectónico e algorítmico. Foi apresentada uma taxonomia possível para processamento paralelo, sendo discutidas as diferenças conceptuais entre as várias categorias apresentadas. Finalmente foi igualmente discutido de que forma o paralelismo inerente aos algoritmos de CAD pode ser eficientemente aproveitado, dependendo da estrutura do problema e da arquitectura em que o mesmo será implementado.

O terceiro capítulo tratou do ambiente em que o trabalho desenvolvido se enquadra, apresentando-se, de forma concisa, um conjunto de técnicas normalmente utilizadas em simulação eléctrica de circuitos. Foi igualmente introduzida a metodologia de simulação que serviu de base a estes desenvolvimentos e discutido o paralelismo eventualmente explorável. Finalmente foi apresentada a arquitectura do multiprocessador utilizado bem como as possibilidades oferecidas pelo mesmo.

No quarto capítulo descreveram-se em detalhe os algoritmos implementados em cada uma das versões. Estas foram testadas num conjunto variado de circuitos de teste, sendo os resultados obtidos apresentados e analisados no capítulo anterior.

Os resultados obtidos são de uma forma genérica extremamente encorajadores, tanto no que respeita à utilização de métodos explícitos por ajuste de exponencial para a resolução do problema de simulação eléctrica de circuitos, bem como em relação à aceleração desse processamento pela utilização de computadores de processamento paralelo.

Uma clarificação dos aspectos matemáticos deste tipo de algoritmos de aproximação por exponencial, permitirá decerto uma maior divulgação e confiança nas técnicas utilizadas. Da mesma forma o estudo de outras arquitecturas como base para os protótipos desenvolvidos pode contribuir para um aumento do desempenho observado, o que torna mais aliciante a utilização e desenvolvimento de tais ferramentas num ambiente de processamento paralelo

6.2 Possíveis Desenvolvimentos

Não era intenção deste trabalho o desenvolvimento de um produto por si só, mas sim o estudo de um conjunto de técnicas inovadoras, tanto no aspecto numérico como algorítmico. Os resultados obtidos satisfazem plenamente os objectivos propostos, pois são de molde a tornar extremamente interessante a utilização dos protótipos desenvolvidos como ferramentas de CAD de uso geral, uma vez que proporcionam uma aceleração real do processo de simulação.

No entanto é possível considerar desde já alguns melhoramentos a efectuar ao nível dos protótipos apresentados, bem como indicar-se a direcção a tomar no seguimento do trabalho iniciado.

Em termos do trabalho já produzido, os melhoramentos possíveis incluem:

- Ao nível do tipo de aproximações numéricas utilizadas:

  - Estudo mais aprofundado das características de convergência e estabilidade dos algoritmos.
  - Determinação de um processo de avaliação do erro cometido em cada instante com a utilização de uma aproximação explícita do tipo exponencial.

- Ao nível da versão baseada em discretização temporal:
  - Transporte da mesma para uma máquina com uma arquitectura com um número de processadores mais elevado, de forma a testarem-se de forma mais concreta os limites de utilização do algoritmo desenvolvido.
  - Estudo de processos de distinção automática entre zonas do circuito em actividade e zonas estáveis de forma a libertar recursos para outras tarefas.

- Ao nível da versão baseada em Partição de Dados:
  - Estudo e determinação de um processo óptimo de escolha da dimensão de um subcircuito face às capacidades disponíveis e à dimensão do circuito global.
  - Estudo e desenvolvimento de algoritmos eficientes de partição dinâmica de um circuito.
  - Conclusão do trabalho iniciado com o algoritmo de partição de dados explorando paralelismo de grão grosso, nomeadamente passando pela sua implementação em arquitecturas do tipo distribuído.

Em termos de pesquisa futura no seguimento do trabalho realizado, o número de opções é praticamente ilimitado. Por um lado torna-se clara a necessidade de desenvolvimento de um conjunto de ferramentas de simulação construído em torno do simulador **CINNAMON** e integrado num ambiente automatizado de projecto de circuitos integrados. Nesse sentido é importante o desenvolvimento de um sistema que possibilite uma simulação interactiva, com rápida visualização dos resultados obtidos e ligado à base de dados contendo a descrição de um dado projecto. Desta forma torma-se viável a alteração automática da configuração ou características de um projecto, de acordo com os resultados obtidos por simulação.

Em termos de Arquitecturas de Processamento Paralelo, este trabalho vem permitir uma primeira apreciação das possibilidades oferecidas pelo conceito de processamento paralelo como meio de aceleração do processo de desenho e verificação de um dado projecto. Neste sentido apontam-se como possíveis seguimentos para este projecto:

- Definição das características arquitectónicas essenciais para a exploração do paralelismo inerente a um problema com uma estrutura irregular, como é o caso da simulação eléctrica de circuitos.

- Desenvolvimento de algoritmos explorando paralelismo ao nível funcional ou combinando paralelismo funcional com paralelismo de dados. Estes podem ser, numa primeira fase, desenvolvidos sobre uma arquitectura do tipo *Alliant*, uma vez que podem ser utilizados os mecanismos existentes para sincronização de tarefas.

- Estudo e desenvolvimento de algoritmos explorando paralelismo por partição ao nível dos dados e aplicáveis a arquitecturas do tipo distribuído. Neste caso é particularmente interessante o estudo de algoritmos de mapeamento de grafos não dirigidos a estruturas de processamento distribuído.

- Exploração de paralelismo ao nível de outros algoritmos envolvidos no processo de desenho de um circuito integrado.

Estes são, no nosso entender, os passos a seguir na direcção encetada por este trabalho.

Apêndice A

Interface com o Utilizador

Neste apêndice apresenta-se o manual de utilizador do programa Cinnamon que pode ser consultado pelos utilizadores do sistema. A sua consulta é feita através do comando do UNIX **man** seguido do nome do programa. O texto encontra-se em inglês por se ter estabelecido esta língua para elaborar toda a documentação técnica relativa ao projecto.

NAME

cinnamon - electric level circuit simulator

SYNOPSIS

cinnamon [-lnstf] circuit_file ...

OPTIONS

The options that can be specified have the following meaning:

- **-l** Minimize the number of points in the output file. This mode limits the amount of i/o the program does when nodes do not change significantly.
- **-n** Create a file with a correspondence between internal and input node numbers.
- **-s** Display run-time statistics.
- **-t** Use old DC algorithm instead of default one (The default one is faster and uses switch-level -like techniques).
- **-f** Create a flat description of the input circuit (to be used by SPICE).

DESCRIPTION

cinnamon is a simulation program based on an explicit event-driven algorithm. It uses a discretization of the voltage axis, whose value should be selected by the user according to the type of the circuit under analysis, as a tradeoff between efficiency and accuracy. Circuits may contain resistors, capacitors, independent voltage sources and MOS transistors. Although the method could be generalized for any type of circuit, the scope of the current version is limited to MOS digital circuits. The input format is compatible with SPICE input format. The current version uses Nassif current models and the Meyer model for charges or the more elaborate Simmy transistor models. Other transistor models will be available in the following versions.

AUTHOR

The INESC Simulation Project:
Luís Vidigal, Horácio Neto, Isabel Teixeira, Luís Silveira, José Guimarães.

FILES

filename.`net`, filename.`dc`, filename.`trans`

LIMITATIONS

Only DC and Transient analysis are available. Only MOS transistors, Resistors, Capacitors and Voltage Sources (piecewise-linear, pulse-wave or dc) are available. No ploting facilities. Output is different from spice output. A waverform viewer (Anim) is provided. Nodeset card not yet implemented. A carefull reading of the User's Manual is recomended.

EXTENSIONS

Cards were added in order to specify the values of the voltage-discretization step. Case insensitive.

REFERENCES

L. M. Vidigal, S. R. Nassif and S. W. Director,
"CINNAMON: Coupled INtegration and Nodal Analysis of MOs Networks",
23rd ACM/IEEE Design Automation Conference, July 1986

SEE ALSO

spice(1).

DIAGNOSTICS

Still under development. Bugs may appear. Refer to the authors in such (unlikely) cases.

Apêndice B

Pseudo-Código para o Simulador Cinnamon

Neste apêndice apresenta-se o algoritmo básico do simulador **Cinnamon**. O algoritmo é apresentado em pseudo-código, baseado na linguagem de programação "C".

```
simulação()            /* seja DC ou Transiente */
{
    t = 0;
    leitura_do_circuito() ;
    marcação_das_fontes() ;
    while( lista_de_acontecimentos != NULL   &&   t < TempoMáximo) {
        próximo_acontecimento(&nó, &tempo, &tensão) ;
        t = tempo ;
        lineariza_dispositivos(nó) ;
        avalia_perturbação(nó, lista de nós adjacentes) ;
        arquiva_resultados(nó, tempo, tensão) ;
    }
}
```

```
marcação_das_fontes()
{
    LISTA_FONTES *fonte_pt = lista_de_fontes;

    for( ; fonte_pt != NULL ; fonte_pt = fonte_pt→next) {
        marca_fonte_na_lista(fonte_pt);
    }
}
```

```
próximo_acontecimento( nó, tempo, tensão)
NÓ **nó;
double *tempo, *tensão;
{
    LISTA_ACONT *lista_aux = lista_acontecimentos;

    *nó→número = lista_aux→número;
    *tempo = lista_aux→tempo;
    *tensão = lista_aux→tensão;
    lista_acontecimentos = lista_acontecimentos→next;
}
```

```
lineariza_dispositivos(nó)
NÓ *nó;
{
    DEV_LISTA *dev_lista = nó→lista_dispositivos;

    for( ; dev_lista != NULL ; dev_lista = dev_lista→next) {
        lineariza(dev_lista, dev_lista→tipo_dev);
    }
}
```

```
avalia_perturbação(nó, lista_nós_adj)
NÓ *nó;
LISTA_ADJ *lista_adj;
{
    LISTA_ADJ *lista_aux = lista_adj;
    double t;

    for( ; lista_aux != NULL ; lista_aux = lista_aux→next) {
        /* t é o tempo da nova perturbação */
        t = avalia_perturbação_nó(lista_aux→nó);
        marca_nó(lista_aux→nó, t);
    }
}
```

```
arquiva_resultados(nó, tempo, tensão)
NÓ *nó;
double tempo, tensão;
{
    fprintf(ficheiro_saída, "%d %e %e\n", nó→número, tempo, tensão);
}
```

Apêndice C

Pseudo-Código para a Versão Acelerada

Neste apêndice apresenta-se o algoritmo básico da versão acelerada do simulador **Cinnamon**, implementada num *Alliant FX/3*. O algoritmo é apresentado em pseudo-código, baseado na linguagem de programação "C".

```
/* Versão Acelerada do simulador Cinnamon (versão paralela)
   Luís Miguel Silveira, Inesc, Abril 1987 */


simulação()              /* seja DC ou Transiente */
{
     t = 0;
     leitura_do_circuito();
     marcação_das_fontes();
     while( lista_de_acontecimentos != NULL   &&   t < TempoMáximo) {
          próximo_acontecimento(&nó, &tempo, &tensão);
          t = tempo;
          disp = nó→número_dev;
          /* em geral um nó é considerado vizinho dele próprio */
          viz = nó→número_nós;
          concurrent_call(..., lineariza_dispositivos, disp, nó);
          concurrent_call(..., avalia_perturbação, viz, nó);
          arquiva_resultados(nó, tempo, tensão);
     }
}
```

```
marcação_das_fontes()
{
     LISTA_FONTES *fonte_pt = lista_de_fontes;

     for( ; fonte_pt != NULL ; fonte_pt = fonte_pt→next) {
          marca_fonte_na_lista(fonte_pt);
     }
}
```

```
próximo_acontecimento( nó, tempo, tensão)
NÓ **nó;
double *tempo, *tensão;
{
    LISTA_ACONT *lista_aux = lista_acontecimentos;

    *nó→número = lista_aux→número;
    *tempo = lista_aux→tempo;
    *tensão = lista_aux→tensão;
    lista_acontecimentos = lista_acontecimentos→next;
}
```

```
lineariza_dispositivos(proc, nó)
int proc;
NÓ *nó;
{
    /* cada nó tem associado uma lista de dispositivos */
    DEV_LISTA *dev_lista = nó→lista_dispositivos[proc];
    lineariza(dev_lista, dev_lista→tipo_dev);
}
```

```
avalia_perturbação(proc, nó)
int proc;
NÓ *nó;
{
    /* cada nó tem associado uma lista de nós vizinhos */
    LISTA_ADJ *lista_aux = lista_adj[proc];
    double t;

    /* t é o tempo da nova perturbação */
    t = avalia_perturbação_nó(lista_aux→nó);
    lock(&semáforo_lista_acontecimentos);
    marca_nó(lista_aux→nó, t);
    unlock(&semáforo_lista_acontecimentos);
}
```

```
arquiva_resultados(nó, tempo, tensão)
NÓ *nó;
double tempo, tensão;
{
    fprintf(ficheiro_saída, "%d %e %e\n", nó→número, tempo, tensão);
}
```

Apêndice D

Pseudo-Código para Discretização Temporal

Neste apêndice apresenta-se o algoritmo básico da versão baseada em discretização temporal. O algoritmo é apresentado em pseudo-código, baseado na linguagem de programação “C”.

```
/* Efit: simulação paralela de circuitos baseada em Discretização temporal
   Luís Miguel Silveira, Inesc & Mit, Outubro 1988 */


simulação()              /* seja DC ou Transiente */
{
    t = 0;
    leitura_do_circuito();
    marcação_das_fontes();
    while( há_actividade_no_circuito()   &&   t < TempoMáximo) {
        t = tempo;
        disp = circuito→número_de_dispositivos;
        nós = circuito→número_de_nós;
        concurrent_call(..., lineariza_dispositivos, disp);
        concurrent_call(..., avalia_perturbação, nós);
        h = mínimo_intervalo_para_propagação();
        concurrent_call(..., calcula_variação, nós, h);
        arquiva_resultados(tempo);
    }
}
```

```
marcação_das_fontes()
{
    LISTA_FONTES *fonte_pt = lista_de_fontes;

    for( ; fonte_pt != NULL ; fonte_pt = fonte_pt->next) {
        marca_fonte_na_lista(fonte_pt);
    }
}
```

```
int há_actividade_no_circuito()
{
    LISTA_ACTIV *lista = lista_actividade;

    for( ; lista != NULL ; lista = lista→next) {
        if(lista→activo != TRUE) return(FALSE);
    }
    return(TRUE);
}
```

```
lineariza_dispositivos(proc)
int proc;
{
    /* assumindo uma lista global de dispositivos */
    DEV_LISTA *dev_lista = lista_dispositivos[proc];
    lineariza(dev_lista, dev_lista→tipo_dev);
}
```

```
avalia_perturbação(proc)
int proc;
{
    /* assumindo uma lista global de nós que possui igualmente uma estru-
      tura de vector */
    LISTA_ACTIV *lista = lista_actividade[proc];
    double t;

    /* t é o tempo da nova perturbação */
    t = avalia_perturbação_nó(lista→nó);
    if( t > 0.0) {
        lista→intervalo = t;
        lista→activo = TRUE;
    } else {
        lista→activo = FALSE
    }
}
```

```
double mínimo_intervalo_para_propagação()
{
    LISTA_ACTIV *lista = lista_actividade;
    h = INFINITY;

    for( ; lista != NULL ; lista = lista→next) {
        if(lista→activo != TRUE) {
            if(lista→intervalo < h) h = lista→intervalo;
        }
    }
    return(h);
}
```

```
calcula_variação(proc, h)
int proc;
double h;
{
    LISTA_ACTIV *lista = lista_actividade[proc];

    variação_tensão(lista→nó, h);
}
```

```
arquiva_resultados(tempo)
double tempo;
{
    LISTA_ACTIV *lista = lista_actividade;

    for( ; lista != NULL ; lista = lista→next) {
        fprintf(ficheiro_saída, "%d %e %e\n", lista→nó→número, tempo,
          lista→nó→tensão);
}
```

Apêndice E

Pseudo-Código para Partição de Dados

Neste apêndice apresenta-se o algoritmo básico da implementação explorando paralelismo de dados de grão grosso. O algoritmo é apresentado em pseudo-código, baseado na linguagem de programação "C".

```
/* Simulador Eléctrico com Partição de Dados (versão paralela)
   Luís Miguel Silveira, Inesc, Outubro 1988 */



/* Admitindo uma configuração com N processadores */
simulação()              /* seja DC ou Transiente */
{
     t = 0 ;
     concurrent_call(..., determina_número_processadores(), N);
     NúmeroÓptimo = determina_número_óptimo_grupos(N);
     NúmeroGrupos = partição_do_circuito(NúmeroÓptimo);
     marcação das fontes de tensão;
     concurrent_call(..., simula_grupos, N);
}
```

```
marcação_das_fontes()
{
    LISTA_FONTES *fonte_pt = lista_de_fontes;

    for( ; fonte_pt != NULL ; fonte_pt = fonte_pt→next) {
        marca_fonte_na_lista(fonte_pt);
    }
}
```

```
simula_grupos(processador)
int processador;
{
    PROCESSADOR *proc = lista_procs[processador];

    while(lista_de_grupos != NULL   &&   t < TempoMáximo) {
    proc→grupo = primeiro_grupo_lista();
    while(há_acontecimentos(proc→grupo) != NULL   &&   t < Tem-
      poMáximo) {
        lê_próximo_acontecimento(proc→grupo, &nó, &tempo, &tensão);
        if( pode_resolver(proc→grupo, nó, tempo) ) {
            t = tempo ;
            próximo_acontecimento(proc→grupo,   &nó,   &tempo,
              &tensão);
            lineariza_dispositivos(nó);
            avalia_perturbação(nó, lista de nós adjacentes);
            arquiva_resultados(nó, tempo, tensão);
            propaga_perturbação((nó, tempo, tensão);
        } else {
            marca_grupo_na_lista(proc→grupo);
        }
    }
}
```

```
lê_próximo_acontecimento( grupo, nó, tempo, tensão)
int grupo;
NÓ **nó ;
double *tempo, *tensão ;
{
    LISTA_ACONT *lista_aux = grupos[grupo]→lista;

    *nó→número = lista_aux→número ;
    *tempo = lista_aux→tempo ;
    *tensão = lista_aux→tensão ;
}
```

```
próximo_acontecimento( grupo, nó, tempo, tensão)
int grupo;
NÓ **nó ;
double *tempo, *tensão ;
{
    LISTA_ACONT *lista_aux = grupos[grupo]→lista;

    *nó→número = lista_aux→número ;
    *tempo = lista_aux→tempo ;
    *tensão = lista_aux→tensão ;
    grupos[grupo]→lista = grupos[grupo]→lista→next;
}
```

```
lineariza_dispositivos(nó)
NÓ *nó ;
{
    DEV_LISTA *dev_lista = nó→lista_dispositivos ;

    for( ; dev_lista != NULL ; dev_lista = dev_lista→next) {
        lineariza(dev_lista, dev_lista→tipo_dev) ;
    }
}
```

```
avalia_perturbação(nó, lista_nós_adj)
NÓ *nó ;
LISTA_ADJ *lista_adj ;
{
    LISTA_ADJ *lista_aux = lista_adj ;
    double t ;

    for( ; lista_aux != NULL ; lista_aux = lista_aux→next) {
        /* t é o tempo da nova perturbação */
        t = avalia_perturbação_nó(lista_aux→nó) ;
        marca_nó(lista_aux→nó, t) ;
    }
}
```

```
arquiva_resultados(nó, tempo, tensão)
NÓ *nó ;
double tempo, tensão ;
{
    fprintf(ficheiro_saída, "%d %e %e\n", nó→número, tempo, tensão) ;
}
```

Bibliografia

[1058Proposal] Workpackage B, Task B.b, *"Esprit Proposal: Knowledge Based Design Assistant for Modular VLSI Design"*, January 1986.

[Agrawal85] Prathima Agrawal, *"Circuit Partitioning for Hardware Simulation Engines"*, International Conference on CAD, 1985.

[AlliantFX] Alliant Computer Systems Corporation, *"Alliant FX Family"*.

[AlliantC] Alliant Computer Systems Corporation, *"Concentrix C Handbook"*, Part Number 302-00004-B, August 1986.

[Arnold85] Jeffrey Arnold, Christopher J. Terman, *"A Multiprocessor Implementation of a Logic-Level Simulator"*, International Conference on CAD, 1985.

[Asbury85] Ray Asbury et al., *"Concurrent Computers Ideal for Inherently Parallel Problems"*, Computer Design, September 1, 1985.

[Bauer87] Romy Bauer, A. Ng, A. Raghunathan, M Saake, C. Thompson, *"Simulating MOS VLSI Circuits Using Super-Crystal"*, Proc. of 1987 VLSI Conference, North-Holland, 1987.

[Bauer88] Romy Bauer, J. Fang, A. Ng, R. Brayton, *"XPSim: A MOS VLSI Simulator"*, Proc. International Conference on Computer Aided Design, Santa Clara, California, November 1988.

[Blank84] Tom Blank, *"A Survey of Hardware Accelerators Used in Computer-Aided Design"*, IEEE Design and Test of Computers, August 1984.

[Bryant80] Randal E. Bryant, *"An Algorithm for MOS Logic Simulation"*, LAMBDA, 4^{th} Quarter, 1980.

[Catlin85] Gary Catlin, Bill Paseman, *"Hardware Acceleration of Logic Simulation using a Data-Flow Architecture"*, International Conference on CAD, 1985.

[Chazan69] A. Chazan, W. Miranker, *"Chaotic Relaxation"*, Linear Algebra and its Applications, vol. 2, 1969.

[Chua75] Leon O. Chua, Pen-Min Lin, *"Computer-Aided Analysis of Electronic Circuits"*, Prentice Hall, Englewood Cliffs, New Jersey, 1975.

[Cox86] P. Cox, R. Burch, B Epler, *"Circuit Partitioning for Parallel Processing"*, IEEE Int. Conf. on Computer-Aided Design, Nov. 1986.

[Dally] William Dally, Randall Bryant, *"A Special Purpose Processor for Switch-Level Simulation"*.

[Dally86] William Dally, *"Directions in Concurrent Computing"*, Proc. 23^{rd} ACM/IEEE Design Automation Conference, 1986.

[Dally89] William Dally, *"The Jellybean Machine"*, Private communication, Massachusetts Institute of Technology, March 1989.

[Delgado] José C. Delgado, *"A Máquina Incremental"*, Tese de Doutoramento, Fevereiro 1988.

[Denneau82] Monty E. Denneau, *"The Yorktown Simulation Engine"*, Proc. 19^{th} ACM/IEEE Design Automation Conference, 1982.

[Deutsch85] J. T. Deutsch, A. R. Newton, *"A Multiprocessor Implementation of Relaxation-Based Electrical Circuit Simulation"*, Proc. 21^{nd} ACM/IEEE Design Automation Conference, 1985.

[Dongarra87] J. Dongarra, S. Lennart Johnsson, *"Solving Banded Systems on a Parallel Processor"*, Parallel Computing, 5(1&2):219-246, 1987.

[Dumlugol86] Dundar Dumlugol, *"The Segmented Waveform Relaxation Method for Mixed-Mode Simulation of Digital MOS Circuits"*, Phd Dissertation, Katholieke Universiteit Leuven, October 1986.

[Duff86] I. S. Duff, A. M. Erisman, J. K. Reid, *"Direct Methods for Sparse Matrices"*, Clarendon Press, Oxford, 1986.

[Gallivan] K. A. Gallivan, P. Koss, S. Lo, R. A. Saleh, *"A Comparison of Parallel Relaxation-Based Circuit Simulation Techniques"*, center for Supercomputing Research, U. Illinois-Urbana.

[George81] A. George, J. W-H. Liu, *"Computer Solution of Large Sparse Positive Definite Systems"*, Prentice-Hall, Englewood Cliffs, 1981.

[Flynn66] M. J. Flynn, *"Very High Speed Computing"*, Proc. IEEE, 14, 1966.

[Ghosh87] Sumit Ghosh, *"A Distributed Modeling Approach for Simulation and Verification of Digital Designs"*, Proc. First International Conference on Computer Technology, Systems and Applications, May 1987.

[Guimarães87] José M. Guimarães, *"MOS Transistor Models in CINNAMON"*, Esprit 1058 Report, June 1987.

[Hahn85] Winfried Hahn, Kristian Fischer, *"MuSic: An Event-Flow Computer for Fast Simulation of Digital Systems"*, Proc. 22^{nd} ACM/IEEE Design Automation Conference, 1985.

[Hancock86] J. M. Hancock, S. DasGupta, *"Tutorial on Parallel-Processing for Design Automation Applications"*, Proc. 23^{rd} ACM/IEEE Design Automation Conference, 1986.

[Hockney88] R. W. Hockney, C. R. Jesshope, *"Parallel Computers 2"*, Adam Hilger, 1988.

[Hoepken87] Harro Hoepken, *"Vectorization of Circuit Simulator ESPICE for FPS-x64"*, Proc. First International Conference on Computer Technology, Systems and Applications, May 1987.

[Hwang84] Kai Hwang, Fayé A.Briggs, *"Computer Architecture and Parallel Processing"*, McGraw-Hill, 1984.

[Hwang86] Seung H. Hwang, Young H. Kim, A. R. Newton, *"An Accurate Delay Modeling Technique for Switch-Level Timing Verification"*, Proc. of the Design Automation Conference, Las Vegas, Nevada, June 1986.

[Jacob85] Georg K. Jacob et al., *"Direct-Method Circuit Simulation using Multiprocessors"*, International Conference on CAD, 1985.

[Jacobs86] G. Jacobs, D. Pederson, *"An Empirical Analysis of the Performance of a Multiprocessor-based Circuit Simulator"*, Proc. of the Design Automation Conference, Las Vegas, Nevada, June 1986.

[Kim84] Young H. Kim, J. E. Kleckner, R. A. Saleh, A. R. Newton, *"Electrical-Logic Simulation"*, Proc. International Conference on Computer Aided Design, 1984.

[Leigthon] F. Thomson Leighton, *"An Introduction to the Theory of Networks, Parallel Computation and VLSI Design"*, To be published.

[Lewis85] David M. Lewis, *"A Hardware Engine for Analogue Mode Simulation of MOS Digital Circuits"*, Proc. 22^{nd} ACM/IEEE Design Automation Conference, 1985.

[Lin86] Tzu-Mu Lin, Carver A. Mead, *"A Hierachical Timing Simulation Model"*, IEEE Transactions on Computer-Aided Design, vol 5, January 1986.

[Lorin72] Harold Lorin, *"Parallelism in Hardware and Software: Real and Apparent Concurrency"*, McGraw-Hill, 1972.

[Lumsdaine88] A. Lumsdaine, J. White, D. Webber, A. Sangiovanni-Vincentelli, *"A Band Relaxation Algorithm for Reliable and Parallelizable Circuit Simulation"*, Proc. International Conference on Computer Aided Design, Santa Clara, California, November 1988.

[Manuel86] Tom Manuel, *"Parallel Processing: the Pace quickens"*, Electronics, September 1986.

[Mead80] Carver Mead, Lynn Conway, *"Introduction to VLSI Systems"*, Addison-Wesley, 1980.

[Meersch86] Erik Vanden Meersch et al., *"SLOCOP: A Timing Verification Tool for Synchronous CMOS logic"*, Proc. European Solid State Circuits Conference, ESSCIRC'86, Delft, September, 1986.

[Nagel75] L. W. Nagel, *"SPICE2: A Computer Program to Simulate Semiconductor Circuits"*, Electronics Research Lab Report, ERL M520, Univ. of Califórnia, Berkeley, May 1975.

[Neto86i] Horácio Neto, Luís Vidigal, *"A Novel Circuit Simulation Algorithm: Implementation Issues"*, 2º Simposio Electrónica das Telecomunicações, May 1986, Lisboa.

[Neto86ii] Horacio C Neto, Luís M Vidigal, *"A Novel Timing Verification Technique"*, Integrated Circuit Technology Conferenec, Limerick, September 1986.

[Neto87i] Horácio Neto, Luís Vidigal, *"CINNAMON: New Results and Improvements"*, Proc. European Conference on Circuit Theory and Design, Paris, 1987.

[Neto87ii] Horacio C Neto, Luís M Vidigal, *"A Novel Approach for Circuit Simulation: Basic Algorithm and New Results"*, Relatório Interno, INESC, Junho 1987.

[Neto89] Horácio C. Neto, Tese de Doutoramento, em preparação, 1989.

[Newton84] A. R. Newton, S. L. Sangiovanni-Vincentelli, *"Relaxation-Based Electrical Simulation"*, IEEE Transactions on Computer-Aided Design, vol 3, March 1984

[Odryna86i] Peter Odryna, Sani Nassif, *"The ADEPT Timing Simulation Algorithm"*, VLSI Systems Design, March 1986.

[Odryna86ii] Peter Odryna, K. Nazareth, C. Christensen, *"A Workstation-Based Mixed Mode Simulator"*, Proc. of the Design Automation Conference, Las Vegas, Nevada, June 1986.

[Paker83] Y. Paker, *"Multi-Microprocessor Systems"*, Academic Press, 1983.

[Philips] Project 1058 Internal Communication, *"The Philips Experimental Multi-Processor System"*.

[Rutenbar88] Rob Rutenbar, Lecture on *"Parallel Placement Algorithms"*, Tutorial Session, 25^{rd} ACM/IEEE Design Automation Conference, 1988.

[Sadayappan] P.Sadayappan, V. Visvanathan, *"Parallelization and Performance Evaluation of Circuit Simulation on a Shared Memory Machine"*, Dept of Comp. and Inf. Sc., Ohio State University.

[Saleh87] R. Saleh, A. R. Newton, *"An Event-Driven Relaxation-Based Multirate Integration Scheme for Circuit Simulation"*, Proc. Int. Symp. on Circuits and Systems, Philadelphia, Pennsylvannia, May 1987.

[Sasaki83] Fohru Sasaka et al., *"HAL: A Block Level Hardware Logic Simulation"*, Proc. 20^{th} ACM/IEEE Design Automation Conference, 1983.

[Sasaki86] Tohru Sasaki et al., *"HAL II: A Mixed-Level Hardware Logic Simulation Systems"*, Proc. 23^{rd} ACM/IEEE Design Automation Conference, 1986.

[Schendel84] U. Schendel, *"Introduction to Numerical Methods for Parallel Computers"*, Ellis Horwood, 1984.

[Seitz85] Charles L. Seitz, *"The Cosmic Cube"*, Communications of the ACM, January 1985.

[Smart87] David W. Smart, Timothy N. Trick, *"Increasing Parallelism in Multiprocessor Waveform Relaxation"*, International Conference on CAD, 1987.

[Silveira87i] Luís Miguel Silveira, *"Study of Multiprocessor Architectures for a Hardware Accelerator"*, Esprit 1058 Report, January 1987.

[Silveira87ii] Luís Miguel Silveira, *"Multiprocessor Implementation of an event-driven Circuit Simulator: Evaluation of Design Options"*, Esprit 1058 Report, July 1987.

[Silveira87iii] Luís Miguel Silveira, *"A Static Partitioning Tool for Circuit Simulation"*, Esprit 1058 Report, March 1987.

[Silveira88i] Luís Miguel Silveira, F. T. Leighton, *"Course Scribe Work: Hypercube Architectures, Properties and Algorithms"*, course 18.435/6.848, Theory of Parallel and VLSI Computation, Massachusetts Institute of Technology, Outubro 1987.

[Silveira88ii] Luís Miguel Silveira, *"A Parallel Band-Matrix Solver for a Circuit Simulation Problem"*, Internal Report, Massachusetts Intitute of Technology and Thinking Machines Corporation, December 1988.

[Silveira89] Luís Miguel Silveira, João Paulo Silva, *"Parallel Circuit Simulation: Preliminary Results"*, Esprit 1058 Report, Janeiro 1989.

[Teixeira86] Isabel M C Teixeira, *"Macromodeling Procedure for CMOS Logic Devices"*, Internal Report, Inesc, December 1986.

[Teixeira87] Isabel M C Teixeira, J P C Teixeira, L M Vidigal, *"A New Approach for Macromodeling Large MOS Networks"*, Internal Report, Inesc, April 1987.

[Trefethen] Lloyd N. Trefethen, *"Finite-Difference and Spectral Methods"*, To be published.

[Tsao85] David Tsao, Chin-Fu Chen, *"A Fast Timing Simulation for digital MOS circuits"*, International Conference on CAD, 1985.

[Uno85] Hiroshi Uno et al., *"A Parallel Implementation of MOS Digital Circuit Simulation"*, International Conference on CAD, 1985.

[Varga62] R. Varga, *"Matrix Iterative Analysis"*, Prentice Hall, Englewood Cliffs, New Jersey, 1962.

[Vidigal86] Luís Vidigal et al., *"CINNAMON: Coupled INtegration and Nodal Analysis of MOs Networks"*, Proc. 23^{rd} ACM/IEEE Design Automation Conference, 1986.

[Vlach83] Jiri Vlach, Kishore Singhal, *"Computer Methods for Circuit Analysis and Design"*, Van Nostrand Reinhold, Berkshire, England, 1983.

[Yeh88] David C. Yeh, Vasant B. Rao, *"Partitioning Issues In Circuit Simulation on Mutiprocessors"*, Proc. International Conference on Computer Aided Design, Santa Clara, California, November 1988.

[Wallich86] Paul Wallich, Glenn Zorpette, *"Minis and Mainframes."*, IEEE Spectrum, January 1986.

[Wallich87] Paul Wallich, Glenn Zorpette, *"Minis and Mainframes."*, IEEE Spectrum, January 1987.

[Wang81] H. H. Wang, *"A Parallel Method for Tridiagonal Equations"*, ACM Transactions on Math. Softw., 1981, 7, pp170-183.

[Webber87] D. M. Webber, A. Sangiovanni-Vincentelli, *"Circuit Simulation on the Connection Machine"*, 24th ACM/IEEE Design Automation Conf., June 1987.

[Weeks73] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quasemzadeh, T. R. Scott, *"Algorithms for ASTAP - A Network Analysis Program"*, IEEE Trans. on Circuit Theory, pp. 628-634, Nov. 1973.

[White85i] Jacob White et al., *"Accelerating Relaxation Algorithms for Circuit Simulation using Waveform Newton, Iterative step size Refinement and Parallel Techniques"*, International Conference on CAD, 1985.

[White85ii] Jacob K. White, A. L. Sangiovanni-Vincentelli, *"Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation"*, International Symposium on circuits and Systems, 1985.

[White86] J. K. White, A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*, Kluwer Pub., Boston, 1986.

[Wiley87] Paul Wiley, *"A Parallel Architecture comes of age at last"*, IEEE Spectrum, June 1987.

[Wing80] O. Wing, J. W. Huang, *"A Computation Model of Parallel Solution of Linear Equations"*, IEEE Trans. on Computers, pp. 632-638, July 1980.