Arquitetura Policíclica Assíncrona

Geraldo Lino de Campos

Resumo

A arquitetura policíclica assíncrona é uma evolução da arquitetura VLIW, caracterizada pela possibilidade de execução simultânea de várias iterações de uma mesma malha, divisão das unidades funcionais em grupos com controle independente, execução antecipada e independência entre o início e o término das operações de acesso à memória. Este conjunto de características permite ao mesmo tempo obter um nível de desempenho superior, hardware simplificado e instruções bastante curtas. Apresenta-se a descrição geral da arquitetura, com ênfase na evolução dos conceitos que levaram à sua definição, e um exemplo de implementação (Projeto Ômicron), atualmente em desenvolvimento na Universidade Estadual Paulista (UNESP).

Abstract

The asynchronous polyciclic architecture (APA) is an evolution of VLIW, characterized by the possibility of simultaneous execution of several loop iterations, division of the functional units in groups with autonomous control, eager execution and decoupled memory access. This properties allow better performance, simpler hardware and short instructions. The APA architecture, the sequence of conceptual phases in its evolution and a practical implementation (Omicron Project), being developed at the São Paulo State University (UNESP), are presented.

Geraldo Lino de Campos é professor associado do Departamento de Engenharia de Computação da Escola Politécnica da Universidade de São Paulo, e responsável pelo Projeto Ômicron na Universidade Estadual Paulista. Áreas de interesse: arquitetura de computadores, análise de desempenho,software básico e processamento de voz e de linguagens naturais.

Endereço: Instituto de Física Teórica da UNESP
Rua Pamplona, 145
São Paulo 01405 - BRASIL
telefone: (55)(11)288-5643
e-mail: GLCAMPOS@BRUSP.BITNET

Arquitetura Policíclica Assíncrona

1 - Introdução

A arquitetura policíclica assíncrona foi desenvolvida para viabilizar a construção de um supercomputador em um ambiente universitário, com as restrições de recursos típicas desta situação: sem possibilidade de utilizar circuitos dedicados e evitando-se empacotamentos e refrigeração exóticas. Restrito ao uso de circuitos comerciais e empacotamentos usuais, não seria possível chegar ao nível de desempenho dos supercomputadores comerciais com uma simples cópia da arquitetura - o que, de qualquer forma, retiraria muito do valor científico do projeto.

Foi então adotada a estratégia de realizar um estudo crítico das arquiteturas existentes, visando determinar evoluções que pudessem remover os aspectos responsáveis por reduções de desempenho, de forma a obter um bom resultado em aplicações gerais, apesar do uso de uma tecnologia inferior.

O problema mais importante na definição de uma arquitetura é sua adequação ao tipo de processamento em que será empregada. Cabe então perguntar: o que caracteriza os processamentos numericamente intensivos, típicos da aplicação dos supercomputadores? Como avaliar o desempenho para as aplicações reais?

Já que não existe uma resposta simples para estas perguntas é necessário adotar uma resposta arbitrária, porém conveniente. Entre os critérios de medida de desempenho de supercomputadores, um dos mais importantes é o LLK (Lawrence Livermore Kernels)[Mah86]. É um conjunto de 24 núcleos (malhas mais internas, onde o programa gasta a maior fração de seu tempo de execução) extraídos de programas reais, escolhidos dentre os que consumiam mais tempo de processador no Laboratório Lawrence Livermore. Como é constituído apenas por núcleos de programas, sua compilação manual é fácil e permite a avaliação de alternativas de arquitetura antes do esforço de escrever um compilador. No âmbito deste trabalho, foram excluídos os núcleos 14 e 22 para eliminar a necessidade de funções como EXP e LN.

Considere-se, por exemplo, o desempenho do Cray-XMP. Para o caso de malhas grandes, seu desempenho varia de 3,4 a 194 MFlops, conforme o núcleo, com uma média geométrica de apenas 33 MFlops. A média geométrica é considerada como a melhor medida central do desempenho, que varia, para aplicações práticas, entre a média harmônica e a média geométrica. A variação do desempenho é de quase duas ordens de grandeza!

A causa deste comportamento é simples, e é a mesma que originou o conceito de arquitetura RISC: se uma máquina possui instruções muito poderosas e específicas, terá um desempenho muito bom quando estas instruções puderem ser utilizadas; quando o programa exigir uma função ligeiramente diferente das previstas na arquitetura, não há como obter uma seqüência de instruções eficiente, e o desempenho será bastante fraco. Este fato é mais grave quando se consideram as arquiteturas vetoriais,

já que existe uma grande classe de processamentos numericamente intensivos que não é vetorizável no sentido das arquiteturas vetoriais convencionais.

Isto é uma conseqüência do fato da arquitetura vetorial só ser adequada ao processamento de algumas das aplicações computacionalmente intensivas. Esta conclusão sugere que deve ser possível obter um melhor resultado nas aplicações práticas, mantendo-se a capacidade de pico, através de uma arquitetura que permita a "vetorização" de uma classe maior de problemas.

Em resumo, o que esta arquitetura pretende é implementar um conjunto de operações que permita processar eficientemente *malhas contendo operações arbitrárias*, incluindo recorrências e operações condicionais, e não apenas as *construções vetoriais*, muito mais restritas em escopo.

Isto é conseguido através de uma combinação de recursos arquitetônicos, utilizando recursos e conceitos derivados da arquitetura VLIW, com extensões para facilitar a execução de malhas e manipulação de grandes seqüências de dados, através da operação independente de grupos de unidades funcionais e de execução antecipada (eager execution).

A arquitetura foi também projetada para eliminar os dois principais problemas de implementação da arquitetura VLIW: a necessidade de um register file com um número muito elevado de portas de escrita e uma instrução muito longa. A implementação é possível com um register file de apenas duas portas de escrita, e instruções com 64 bits.

2 Considerações gerais sobre arquiteturas

A taxonomia dos computadores atuais é bastante complexa [Ski88], [Dun90]; entretanto, pode-se destacar, de uma forma bastante simplificada, as seguintes famílias de arquiteturas:

2.1 Processadores SISD convencionais

Considere-se o caso de um processador convencional, SISD (Single Instruction stream/Single Data stream) e pipelined. Esta família de arquiteturas não explora nenhum tipo de paralelismo.

O desempenho deste tipo de processadores pode ser estimado por

$$D = \frac{1}{N \times C \times T,}$$

onde N é número de instruções a executar, C é o número médio de ciclos por instrução, e T é o tempo de ciclo. N e C dependem da arquitetura do processador; assim, processadores CISC reduzem N melhorando a funcionalidade das instruções, ao preço de aumentar o valor de C; os processadores RISC procuram obter um valor de C próximo a 1, aceitando um aumento no valor de N.

O valor de T é essencialmente limitado pela tecnologia de produção de microcircuitos, embora seja também influenciado pela arquitetura, já que uma estrutura de controle simples permite obter, com uma mesma tecnologia, um valor de T menor que o de uma arquitetura com uma estrutura de controle mais complexa.

É importante lembrar que um processador executa dois tipos de instruções: instruções úteis, que efetivamente realizam a atividade pretendida pelo usuário, e instruções de serviço, que preparam a execução das instruções úteis. Assim, o comando

```
A(I) = B(J) + C (K)
```

executa uma instrução útil: realiza uma soma. Para realizar esta operação, é obrigado a realizar um certo número de instruções de serviço; seu número é variável com o conjunto de instruções adotado, e da presença ou não dos valores necessários em registradores. Para uma máquina com arquitetura RISC pura e todos os valores referidos ainda na memória, pode-se ter 12 instruções de serviço no exemplo acima; uma máquina CISC poderá eventualmente ter 3. Uma medida mais adequada de desempenho pode ser obtida excluindo-se as instruções de serviço, como será feito adiante; nesta seção será utilizado o número bruto de instruções executadas, como uma primeira aproximação.

Fixado T, a obtenção de um desempenho mais elevado exige a adoção de arquiteturas que explorem alguma forma de paralelismo na execução das instruções; as outras famílias de arquiteturas apresentadas a seguir foram desenvolvidas com este propósito.

2.2 Processadores SISD pipelined

Nesta família de arquitetura, as instruções são processadas seqüencialmente, uma por uma, passando por uma série de estágios de duração T, isto é, a execução da instrução é segmentada em uma série de subprocessos consecutivos, cada um executado em um estágio do pipeline. As instruções entram e saem de cada estágio na mesma ordem em que foram especificadas na seqüência do programa. Esta família de arquiteturas explora apenas o paralelismo temporal no processo de execução das instruções.

A detecção e a resolução de conflitos por dados ou instruções é geralmente detectada pelo hardware, embora possam também ser detectadas e resolvidas pelo software.

Neste caso, o número de ciclos por operação, no que concerne ao desempenho, fica reduzido a 1, embora a duração individual de uma operação possa ser de muitos ciclos.

$$D = \frac{1}{N \times 1 \times T}$$

É importante destacar que a estimativa acima refere-se ao máximo que pode ser obtido; a existência de operações de desvio e de dependências entre dados e resultados pode causar uma redução significativa no desempenho, como se verá na próxima seção. Isto ocorre porque, ao contrário da arquitetura SISD não pipelined, o processador pode ter de parar para que o pipeline seja preenchido após um desvio ou para esperar o resultado de uma operação ainda em progresso. Estas interrupções do fluxo do pipeline são chamadas de bolhas.

A literatura dá muita ênfase ao problema das bolhas resultantes de operações de desvio, mas, pelo menos no caso dos processamentos numericamente intensivos, este problema é muito menos importante que o das bolhas derivadas da necessidade de esperar por resultados. Este último problema será detalhadamente estudado na próxima seção.

2.3 Processadores superescalares pipelined

Os processadores superescalares visam explorar o paralelismo espacial além do paralelismo temporal, através do uso de múltiplas unidades funcionais, e de um controlador com capacidade de despachar várias instruções por ciclo.

O desempenho máximo que pode ser obtido para esta família de arquiteturas é

$$D = \frac{F}{N \times 1 \times T}$$

onde F é o número de unidades funcionais presente numa particular implementação.

Novamente, D é um número máximo, que depende não só da presença de bolhas nos pipelines como da existência de paralelismo na seqüência de instruções a executar. O nível de paralelismo estimado num programa típico é bastante variável conforme a fonte; a referência [Jou89] determinou valores entre 1,6 para um programa essencialmente seqüencial até 6, para algumas malhas numericamente intensivas e um compilador muito sofisticado. A limitação do paralelismo decorre da dependência entre dados e resultados.

2.4 Processadores pipelined com instruções muito longas

Os processadores pipelined com instruções muito longas - IML (Very Long Instruction Word - VLIW) também possuem múltiplas unidades funcionais, mas seu princípio de funcionamento é radicalmente diferente. Em lugar de despachar várias instruções em um mesmo ciclo, a instrução especifica individualmente as operações a serem realizadas por cada uma das unidades funcionais. As unidades funcionais podem ser de tipos distintos, especializadas por função. A instrução resultante torna-se bastante longa (tipicamente, de 256 a 512 bits), daí o nome da arquitetura.

Em um processador com arquitetura IML, cada campo da instrução é decodificado em paralelo, e cada operação é iniciada na respectiva unidade funcional. Os compiladores devem ter conhecimento dos detalhes da implementação da arquitetura, e são responsáveis pela determinação da ordem da execução das instruções, pela obediência aos tempos de latência e por outros detalhes que devem ser cuidados pelo hardware nas arquiteturas superescalares.

Assim, não há necessidade de utilizar mecanismos de controle de travamento do pipeline, ou de controle de emissão de instruções. Além disto, como as unidades funcionais podem ser dedicadas a operações específicas (por exemplo, cálculo de endereços, realização de desvios, soma em ponto flutuante, multiplicação em ponto flutuante), podem ser mais simples. Estas simplificações do hardware permitem que se obtenha ciclos de máquina mais curtos que nos outros casos.

Um processador IML puro permite que todas as unidades funcionais operem ao mesmo tempo. As implementações reais costumam estabelecer compromissos quanto a esta simultaneidade, por limitação quanto ao número de acessos simultaneos que podem ser feitos a um register file. Analogamente, é usual fazer compromissos quanto ao formato da instrução, já que em um grande número de casos,a maior parte das subinstruções especifica que a respectiva unidade funcional fique inativa.

Esta família de arquiteturas também explora os paralelismos espacial e temporal, e apresenta o mesmo desempenho máximo de um processador superescalar:

$$D = \frac{F}{N \times 1 \times T}$$

onde F é o número de unidades funcionais. É preciso, entretanto, interpretar adequadamente o significado de F, já que as unidades funcionais podem ser específicas para cada função. Assim, este desempenho máximo só será atingido quando houver a possibilidade de usar todas as unidades funcionais existentes. As considerações sobre o paralelismo disponível no programa são igualmente válidas, cabendo acrescentar que se aplicam agora a cada tipo de unidade funcional, se estas forem diferentes.

Como no caso dos processadores superescalares, os valores reais obtidos para D são bem menores que o máximo indicado pela fórmula acima.

2.5 Processadores com Arquitetura Policíclica Assíncrona

A arquitetura policíclica assíncrona descrita neste relatório deriva da arquitetura IML. As principais evoluções consistem em:

- utilização da variante policíclica da arquitetura IML;
- desacoplamento entre o início de operações de leitura na memória e utilização do valor lido;
- execução antecipada seletiva;
- utilização de um registrador de iteração e controle automático de repetição de malhas;
- operação assíncrona de subconjuntos dos elementos das unidades funcionais.

Assim, seu desempenho máximo é o mesmo dos processadores IML:

$$D = \frac{F}{N \times 1 \times T}$$

Entretanto, seu desempenho médio pode ser consideravelmente superior, já que a operação assíncrona permite uma série de alternativas de programação para os casos que envolvem dependências entre dados e dependências de controle.

3 Arquitetura policíclica

A arquitetura policíclica é baseada na variante chamada policíclica [Rau82] ou de fluxo de dados orientado [Rau89] da arquitetura PIML. Seu conceito essencial é permitir a execução concorrente

de várias iterações de uma mesma malha. O nome policíclico foi conservado de [Rau82a] por ser significativo, embora a estrutura de hardware e o controle do prólogo e epílogo de malhas seja radicalmente diferente dos apresentados nas referências acima.

Várias construções usuais em linguagens de programação possuem recursos de hardware especiais que suportam sua execução. Assim, a recursão é suportada por algum esquema de apontadores que permite desambiguar as referências feitas a um mesmo elemento em níveis diferentes de recursão. A arquitetura policíclica possui um elemento análogo, chamado apontador de iteração, destinado a permitir que um mesmo trecho de código se refira a iterações paralelas distintas, desambiguando referências ao conjunto de registradores.

Sua implementação é muito simples. Referências a registradores podem ser absolutas ou dinâmicas. O primeiro caso corresponde à situação convencional; no caso de uma referência a um registrador dinâmico soma-se o conteúdo do apontador de iteração ao número do registrador referido na instrução. Uma particular instrução de desvio, utilizada para indicar o fim textual de uma malha e desvio para seu início, tem a propriedade de incrementar o valor deste apontador, iniciando uma nova iteração. O incremento é variável, de forma a abranger todos os valores intermediários necessários para a execução completa de uma iteração.

Como o apontador de iterações é incrementado no início de cada nova iteração, cada iteração acessa um conjunto distinto de registradores físicos, e qualquer resultado computado durante a iteração corrente ou qualquer das anteriores (até o limite determinado pelo tamanho do conjunto de registradores módulo incremento da iteração) pode ser obtido utilizando-se um desvio apropriado a partir do valor corrente do apontador de iterações.

Esta arquitetura é de certa forma dual da arquitetura vetorial. Na arquitetura vetorial, lê-se um par de vetores da memória principal, armazenam-se estes vetores em registradores, e se inicia uma operação múltipla sobre seu conteúdo. Na arquitetura policíclica, lêem-se um a um os elementos de todos os vetores necessários para uma iteração, aplicando-se sobre eles as operações necessárias. Como estas operações são arbitrárias, e os valores correspondentes às iterações anteriores estão disponíveis em registradores, é possível "vetorizar" uma classe muito maior de programas.

O nome de arquitetura policíclica aplica-se a qualquer arquitetura que apresente esta propriedade. As implementações descritas em [Rau82a] e em [Rau89] são bastante diferentes entre si, e diferentes também da implementação proposta, particularmente no que diz respeito à execução condicional e à execução de prólogos e epílogos de malhas.

4 - Arquitetura policíclica assíncrona

Esta arquitetura unifica e extende os conceitos de arquitetura policíclica e de arquitetura VLIW. Sua evolução se inicia por uma análise crítica da arquitetura IML, já ampliada pelo conceito de execução policíclica. Suas características derivam da necessidade de obter uma arquitetura de maior desempenho e com implementação viável com componentes normais.

Um processador IML é caracterizado por:

1. Um único fluxo de execução;
2. Um número grande de unidades funcionais, tanto para execução de operações aritméticas quanto para movimentação de dados, todas com controle planejado em tempo de compilação;
4. Possuir operações que utilizem um número de ciclos pequeno e predizível estaticamente;
5. Todas as operações podem ser pipelined, isto é, cada unidade funcional pode iniciar uma nova operação em cada ciclo, independentemente do número de ciclos que a operação necessite para ser completada.

Um processador IML ideal deve ter muitas unidades funcionais ligadas a um grande conjunto de registradores central. Cada unidade funcional pode ter várias portas de leitura (duas para as unidades aritméticas) e pelo menos uma porta de escrita, todas ligadas ao register file, que deve ter um número de

portas suficientemente grande para suportar os pedidos de todas as unidades funcionais sem que haja conflito de endereçamento.

Esta descrição leva a um register file com um número de portas irrealista, e, como o nome indica, a uma instrução muito longa. Assim, os projetos reais tem que aceitar compromissos em alguns dos requisitos. O problema mais importante é o do número de portas de escrita no register file, já que todas as unidades funcionais devem ter a capacidade de escrever, em um mesmo ciclo devido às operações serem pipelined, o resultado de sua operação (o problema do número de portas para leitura também existe, mas é menos grave, pois os registradores podem ser fisicamente replicados o número necessário de vezes). Em lugar de tentar uma solução ad hoc, como em outras implementações, convém proceder a uma análise detalhada dos requistos acima, modificando-os para obter melhor desempenho e execução física viável, sem comprometer as boas características que apresenta.

4.1 Operação independente das unidades de endereço

A primeira impossibilidade real diz respeito à condição 4. Uma classe de unidades funcionais, das mais importantes para o desempenho, que é a das unidades encarregadas do acesso à memória, não pode atender ao requisito de preditibilidade do número de ciclos necessários para sua operação. Esta condição não pode ser satisfeita pelo processo de acesso à memória, já que o tempo real de acesso vai depender do endereço particular referenciado e de outros acessos pendentes.

O modelo pode ser mantido adotando-se o conceito de tempo virtual. Se um dado for necessário para a execução de uma instrução e ainda não estiver disponível, o relógio do processador é parado até que o dado se torne disponível. Esta solução exige uma estimativa precisa do tempo de acesso; se a estimativa for muito pequena, o processador ficará muito tempo parado; se for muito grande, haverá desperdício de ciclos apesar do dado já estar disponível. Nos dois casos o desempenho é comprometido.

Esta restrição quanto ao número de ciclos para acesso à memória pode ser levantada através do desacoplamento entre as operações de início de acesso à memória e ao uso dos valores resultantes deste acesso. Assim, ficam criadas unidades de endereço, com a função de enviar endereços ao subsistema de memória, e unidades de acesso, de onde efetivamente os valores presentes na memória são obtidos. O funcionamento composto destes componentes é o seguinte:

O processador, através das unidades de endereço, envia para o subsistema de memória o endereço das variáveis que vai necessitar. O subsistema de memória atende a estes pedidos, na medida e na ordem que puder, e envia os valores correspondentes às unidades de acesso, que são responsáveis por reordenar os valores e mantê-los em uma fila, de onde podem ser retirados pelas outras unidades funcionais, na mesma ordem em que foram pedidos ao subsistema de memória.

Este esquema permite ignorar o tempo de acesso à memória, desde que exista um número grande de operações pendentes, e que se possa iniciar as operações de leitura da memória com bastante antecedência em relação ao momento em que o valor correspondente se torne necessário. Quando estas condições não puderem ser satisfeitas, pode-se ainda recorrer ao modelo de tempo virtual.

O uso de tempo virtual é particularmente eficaz nos trechos de programa com execução seqüencial, mas leva a soluções que não são ótimas na execução de malhas; pelo contrário, leva a um tempo de execução que é o resultante da soma dos piores tempos que podem ocorrer. Isto decorre principalmente do fato de que, quando o processador pára para esperar um acesso, deixa também de formular novos pedidos de acesso, reduzindo o número de acessos pendentes e o nível de paralelismo no acesso à memória.

Um segundo inconveniente de manter os comandos para geração de endereço na mesma instrução é que bits de instrução são gastos de uma forma repetitiva, já que neste caso os endereços a acessar formam sempre uma progressão aritmética. O mesmo pode ser dito sobre o uso de portas do register file.

É possível, entretanto, obter um tempo de execução que é proporcional à média dos tempos de acesso envolvidos, e sem usar bits de instrução e acessos ao register file, desde que o processo de geração de endereços seja autônomo e assíncrono com a execução das instruções aritméticas durante a

execução de malhas. Assim, este processador se afasta da arquitetura IML eliminando a obediência à condição 1, permitindo a existência de vários *loci* de controle.

Embora a motivação original tenha sido ligada ao problema de geração de endereços, foi possível concluir que este recurso é bastante geral, e oferece uma solução completa e eficiente para os problemas de tamanho da instrução e número de portas do register file. Na verdade, não há nenhuma razão, conceitual ou prática, que justifique a existência da condição 1.

Deixando-se de lado os aspectos práticos de implementação, que serão examinados posteriormente, pode-se supor que cada unidade funcional possui seu próprio cache e seus próprios mecanismos de controle de execução, e que um fluxo de controle principal dispara a execução autônoma de um grupo de unidades funcionais que lhe são subordinadas. Este grupo, por sua vez, passa a ter um fluxo de controle principal, que pode disparar um subgrupo de suas unidades funcionais, e assim sucessivamente.

A arquitetura passa a dispor de dois tipos de unidades de endereço: unidades de endereço sob demanda, para a obtenção de variáveis individuais, e unidades de endereço múltiplo, para a obtenção de conjuntos de variáveis.

As unidades de endereço múltiplo, que possuem operação assíncrona, permitem obter variáveis de vários vetores, com endereços em progressão aritmética. Os valores correspondentes às progressões aritméticas desejadas - valor inicial, incremento, e opcionalmente limite, chamados coletivamente de descritor - são enviados a registradores das unidades de endereço múltiplo, juntamente com o total de elementos a serem gerados. A programação experimental de diversos exemplos mostrou que não há vantagem em utilizar os elementos do register file geral para este fim; é mais conveniente copiá-los para um conjunto de registradores próprio do processador de endereços. Este conjunto de registradores é percorrido seqüencialmente, gerando-se um endereço em correspondência a cada descritor encontrado, até que seja gerado o total de endereços desejado. Evidentemente, este processo será paralisado sempre que o subsistema de memória não puder receber novos pedidos de acesso, e será reiniciado quando esta condição tiver sido eliminada.

Esta forma de operar tem duas conseqüências importantes.

A primeira conseqüência é que esta forma de acesso só é realmente eficaz quando existe um grande número de acessos pendentes. Esta condição praticamente exige a adoção do modelo de execução antecipada (eager execution).

A utilização de execução antecipada obriga as operações de acesso a memória a apresentar um comportamento peculiar. Em execução convencional, um valor só deve ser solicitado à memória quando necessário; no modo de execução antecipada, o valor deve ser solicitado tão cedo quanto possível, sem considerar a possibilidade desse valor vir a não ser necessário, ou de seu endereço só vir a ser validado por testes posteriores. Se o fluxo do programa determinar posteriormente que o valor não é necessário, deve simplesmente ser lido por uma unidade funcional qualquer, que esteja livre no momento adequado, e a seguir ignorado. Como resultado, todas as interrupções ligadas ao acesso à memória (endereço inválido, erro de paridade, etc) devem ser retardadas até que o valor ofensor seja efetivamente utilizado.

Como estas considerações valem também para a execução antecipada de operações com os valores lidos da memória, as interrupções devem ser ainda mais atrasadas: só devem ocorrer quando um registrador marcado como contendo um valor inválido for efetivamente utilizado. O conteúdo de um registrador é efetivamente utilizado quando for armazenado novamente na memória, ou quando for utilizado como elemento de decisão em uma operação de desvio.

Este recurso é implementado pela associação de um descritor de resultado (DR) a todas as palavras escritas no register file. O DR possui dois campos, erro e estado. O valor 0 no campo de erro indica que o conteúdo do registrador é válido. O campo de estado possui uma cópia do registrador de estado do processador que gerou o valor contido nesse registrador.

Caso uma palavra contendo erro com valor diferente de 0 seja utilizada como um operando, a operação não é realizada e o valor de DR é copiado inalterado para o registrador de destino do resultado. Caso os operandos sejam válidos, o campo de estado conterá sempre uma cópia do registrador de estado do processador ao fim da operação; se o resultado for inválido, como no caso da divisão por zero, o campo de erro refletirá esta situação.

Os mecanismos de execução antecipada e de interrupção retardada oferecem uma alternativa de execução que é melhor que trace scheduling ou soluções equivalentes; em lugar de desfazer as operações quando o trace preferencial não for executado, vários traces podem ser executados em paralelo, preservando-se apenas os resultados do trace correto. Obviamente, nada impede a utilização das técnicas de trace scheduling, descritas por exemplo em [Fis81] e [Lam88] quando o presente mecanismo não for apropriado.

Uma segunda conseqüência da operação assíncrona é que se torna bastante complexo especificar um registrador dinâmico como destino de uma operação de acesso à memória. Novamente, a programação experimental mostrou que isto não é um problema; pelo contrário, obtem-se uma menor latência quando o acesso aos dados provenientes da memória são utilizados diretamente pela unidade funcional que dele necessita. Assim, a unidade de acesso à memória é apenas uma fila de onde as outras unidades funcionais podem retirar dados provenientes da memória.

Utiliza-se um mecanismo análogo para as operações de escrita.

4.2 Operação independente de grupos de unidades funcionais

A existência de um grupo independente de unidades funcionais de endereço múltiplo propicia:

- Operação à máxima velocidade possível, independentemente de paralizações de outras unidades funcionais por espera de algum recurso;
- Redução no número do portas do register file, já que as unidades funcionais em operação autônoma escrevem no seu próprio conjunto de registradores;
- Redução no tamanho da instrução, já que cada grupo precisa receber apenas comandos para suas próprias unidades funcionais.

Não há porque restringir a capacidade de operação assíncrona apenas às unidades de endereço múltiplo, já que as vantagens apontadas acima podem ser obtidas também para o caso de serem criados outros grupos.

Assim, pode-se caracterizar um processador APA pela capacidade de poder dividir suas unidades funcionais em vários grupos, cada um capaz de operação autônoma, tendo uma visão autônoma de seu conjunto de registradores e necessitando apenas dos comandos relativos às unidades funcionais que o integrem. Exigem-se, obviamente, instruções que permitam ao processador designar um subconjunto de seus recursos como participante de um grupo para operação autônoma, iniciar sua operação autônoma, e transferir dados e resultados.

Um processador como o assim descrito é certamente factível sem problemas graves de hardware e sem perder a simplicidade que permite um ciclo rápido aos processadores IML. Pode ainda ser mais simplificado a partir das considerações abaixo, resultantes da programação experimental de estruturas típicas:

- Os grupos podem ser fixados a priori; as necessidades de divisão variam muito pouco de caso para caso;
- Os casos em que se pode aproveitar diretamente o conjunto completo de recursos são muito raros, de modo que é possível trabalhar sempre com os grupos já divididos;
- Grupos que envolvam unidades aritméticas necessitam sempre de grupos de unidades de endereço múltiplo;
- A comunicação direta entre grupos é muito reduzida.

Estas considerações permitem uma notável simplificação a nível de hardware. Como cada grupo tem uma visão própria de seu register file, o número de portas de escrita pode ser mantido muito baixo, e como cada grupo precisa receber apenas seus próprios comandos, a instrução pode ser bastante curta. No exemplo de implementação descrito adiante, a instrução pode ter apenas 64 bits, e os register files apenas duas portas de escrita, sem prejuízo de um elevado índice de desempenho.

O fato da instrução formal ser pequena não limita a capacidade de controle, já que o tamanho efetivo da instrução é o resultado da multiplicação do tamanho formal pelo número de grupos em operação

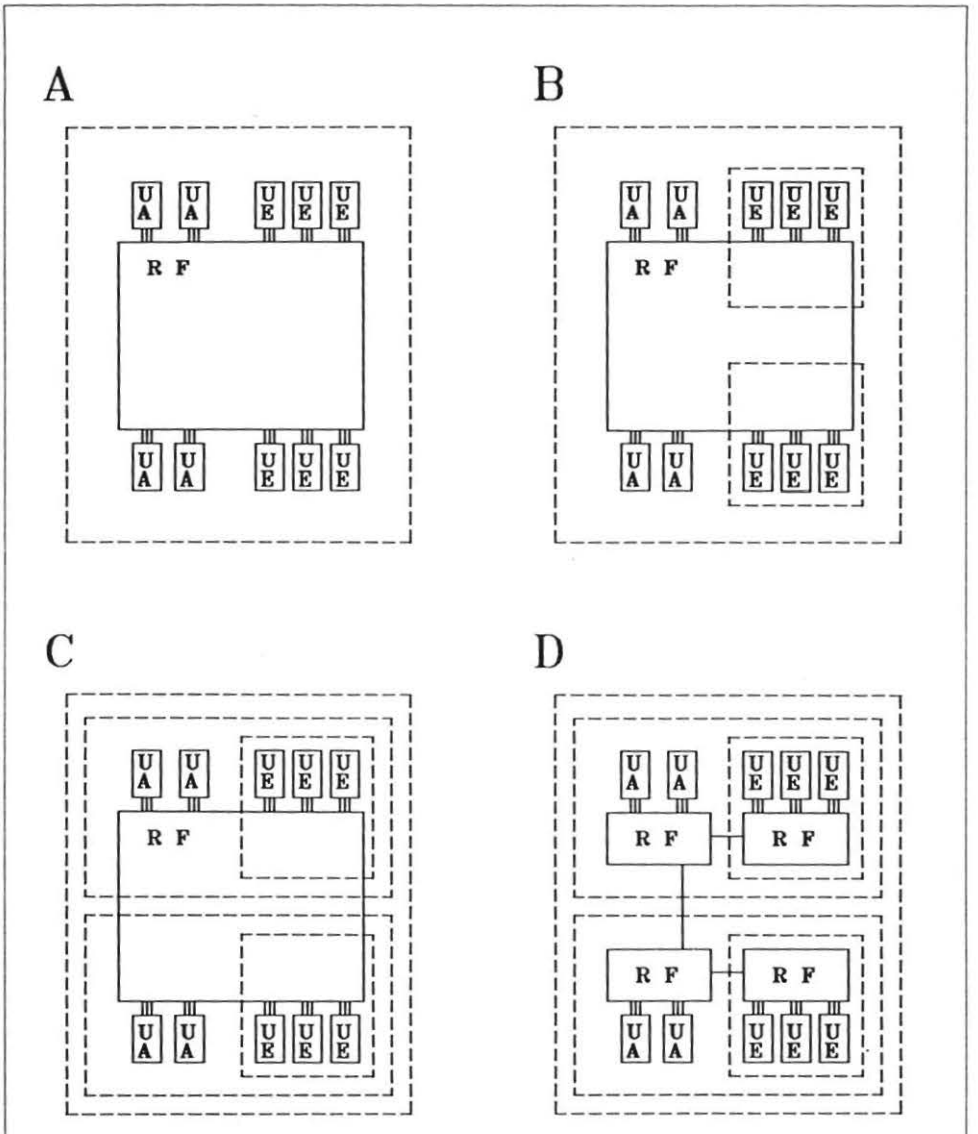

Figura 1 - Evolução da arquitetura IML para APA

Em A, representa-se a arquitetura IML convencional. Em B e C, as unidades de endereço passam a constituir grupos autônomos; em D, a arquitetura APA, sob o ponto de vista do fluxo de controle.

Convencões: UA - unidade aritmética; UE - unidade de endereço; RF - register file. Unidades que apresentam o mesmo fluxo de controle estão envolvidas por linha tracejada. As unidades irrelevantes a esta análise não são exibidas.

assíncrona; existe um ajustamento automático do tamanho efetivo da instrução ao grau de paralelismo existente no trecho de programa em execução.

O número de grupos e de unidades funcionais por grupo deve ser determinado em função da natureza da aplicação e de outras características do processador em projeto, como a capacidade de obter operandos da memória.

4.3 Caracterização da Arquitetura Policíclica Assíncrona

Em resumo, um processador APA é caracterizado por:

1. Um número grande de unidades funcionais, tanto para execução de operações aritméticas quanto para movimentação de dados, todas com controle planejado em tempo de compilação;
2. As unidades funcionais estão divididas em grupos; cada grupo possui um fluxo de execução próprio e uma visão própria do conjunto de registradores sobre o qual atua;
3. Instruções com um número suficiente de bits para controlar direta e independentemente as unidades funcionais de cada grupo;
4. Possuir operações que utilizem um número de ciclos pequeno e predizível estaticamente, em tempo virtual;
5. Todas as operações podem ser pipelined, isto é, cada unidade funcional pode iniciar uma nova operação em cada ciclo, independentemente do número de ciclos que a operação necessite para ser completada;
6. Utilizar execução antecipada e interrupções retardadas.

A figura 1 ilustra a evolução do modelo de arquitetura, de IML para APA.

5 Técnicas de programação

Uma descrição detalhada das técnicas de programação de computadores com APA ultrapassa o escopo deste texto, podendo ser encontrada em [Alv90]; sumariamente, pode-se considerar que a limitação ao paralelismo médio utilizável é devida à dependência entre dados e resultados, isto é, novas operações não podem ser iniciadas por depender de resultados que ainda estão sendo calculados. Como as técnicas de programação explorando o aspecto policíclico da arquitetura são, em linhas gerais, as expostas em [Den89], não serão apresentadas aqui.

Assim, o objetivo desta seção é expor os aspectos envolvendo a exploração do paralelismo, em particular no caso de malhas. Neste caso, as técnicas de programação dependem da natureza das dependências encontradas.

5.1 Ausência de dependências

Neste caso, o paralelismo disponível só é limitado pelo número de repetições da malha e pelo número de acessos simultaneos à memória. Pode ser implementado pela operação assíncrona de um número arbitrário de grupos de unidades funcionais, cada uma calculando um subconjunto das iterações necessárias.

Um exemplo é o cálculo de soma de produtos, onde diversos grupos podem calcular somas parciais; após o término de todas as somas parciais, o grupo que possua o fluxo de controle principal deve receber estas somas parciais e calcular a soma final.

5.2 Dependência de valores em uma mesma execução da malha.

Este é um caso particular da dependência entre operações sucessivas, e o fator que limita o desempenho é a latência das operações. Para reduzir esta latência utiliza-se um registrador interno, eliminando-se da latência o tempo da escrita do resultado no register file.

5.3 - Dependência de valores entre execuções sucessivas da malha

Este é o caso onde é mais difícil explorar o paralelismo no caso geral. Possui influência muito grande no desempenho no caso de malhas envolvendo poucas operações por repetição. Neste caso, existe uma solução parcial, que consiste em aumentar o passo da iteração, transformando a malha de passo n em m malhas de passo n*m. Isto aumenta o número de operações por iteração, mas, com as novas malhas distribuidas por grupos distintos, pode haver um sensível aumento no desempenho.

O exame de casos reais mostra que é possível obter até o dobro do desempenho original para m = 2, e que para m > 2 o aumento de desempenho é bem mais raro, devido ao grande aumento no número de operações que acarreta.

6 - Dimensionamento de uma implementação da arquitetura policíclica assíncrona

Esta seção apresenta e justifica os principais aspectos de projeto adotados para o computador Ômicron, objeto do presente projeto.

O projeto Ômicron preve o desenvolvimento de um processador em tecnologia ECL, com tempo de ciclo de 10 ns, e elevado índice de desempenho para processamentos numericamente intensivos típicos. Utiliza a arquitetura APA.

6.1 - Aspectos gerais

Para simplificar o projeto e sua implementação, tanto em hardware como em software, todas as operações de entrada e saída serão realizadas por um processador independente. Esta solução permite também otimizar o uso do processador de dados, que é projetado para a execução de operações numericamente intensivas e não para as funcões típicas de controle de dispositivos periféricos, redes, etc. Em particular, sendo um processador intensivo no uso de registradores, tem um tempo de comutação entre tarefas bastante alto, o que o torna ineficiente para o tratamento de uma taxa de interrupções elevada.

As arquiteturas de propósito geral baseiam-se quase invariavelmente em caches para melhorar o desempenho de execução, aproveitando a validade quase universal do princípio da localidade. Embora o princípio da localidade seja válido para programas de natureza geral, é falso para processamentos numericamente intensivos, pelo menos para qualquer tamanho realista do cache. Esta conclusão foi estabelecida para várias máquinas, de diferentes arquiteturas [Die88], [Col88], [Rau89].

Neste caso, o que se necessita é um nível elevado de entrefolhamento na memória; o tempo de acesso pode ser alto, mas desde que em um certo instante exista um número adequado de pedidos de acesso sendo processados, pode-se obter operandos com a velocidade necessária. O acesso à memória é certamente o aspecto mais importante para o desempenho de um supercomputador.

O subsistema de memória ideal deve oferecer grande capacidade, tempo de acesso baixíssimo e banda passante muito alta, tudo isto com baixo custo. As memórias reais não podem atender a todos estes requisitos e precisam sacrificar alguns dos objetivos.

No caso presente, em que não há como justificar o custo de memórias estáticas, torna-se obrigatório o uso de memórias dinâmicas, de maior capacidade e baixo custo. Durante a execução de malhas o entrefolhamento torna irrelevante o tempo de acesso, desde que o tempo entre inícios de operação possa ser mantido suficientemente curto; nas partes seqüenciais de programas pode haver problemas de desempenho, mas felizmente, no caso da APA, estes trechos não são importantes para o desempenho final, já que praticamente todas as malhas podem ser "vetorizadas".

Sistemas dedicados a processamentos numéricos intensivos usualmente não apresentam recursos de memória virtual.

O processador proposto admite memória virtual, mas não tem mecanismo de suporte a falhas de página. Assim, toda a área a ser referida por um programa deverá estar mapeada para a memória real antes do início de sua execução.

Tabela 1 - Desempenho para várias alternativas de grupos e unidades funcionais (desempenho em megaflops)

Arquitetura	Clock	Grupos	Unidades aritméticas por grupo	Mínimo	Média harmônica	Média geométrica	Média aritmética	Máximo
Vetorial CRAY X-MP	8.5	-	-	3	16	33	65	194
APA	10	1	1	16	42	48	49	97
APA	10	1	2	20	61	84	98	187
APA	10	1	3	20	64	94	118	276
APA	10	1	4	20	64	97	127	355
APA	10	2	2	20	75	115	148	371
APA	10	4	1	16	59	86	117	355

A memória virtual é implementada através da utilização de páginas de 8 MBytes, com mapeamento direto. O número de páginas que deve ser suportado por processador deve ser proporcional à memória máxima prevista para uma determinada implementação.

6.2 Dimensionamento das unidades funcionais

Nesta fase, é importante dimensionar os recursos para que o processador possa atender eficientemente às situações reais, sem preocupação com o nível de desempenho de pico que poderia ser atingido. Os LLKs foram utilizados como programas típicos.

Os resultados apresentados a seguir são resultado de simulações realizadas sobre várias alternativas de arquitetura, com os vínculos resultantes da possibilidade de sua execução física. Os detalhes podem ser encontrados em [Alv90].

A primeira consideração é o número de caminhos de acesso à memória. Em teoria, é possível incorporar um número arbitrário de unidades funcionais a um processador APA; entretanto, decididamente não é possível implementar um número elevado de caminhos de acesso à memória utilizando uma chave crossbar realista. Assim, a primeira consideração deve ser o número de operações simultâneas de acesso à memória. Para evitar um número mínimo de submódulos muito alto, bem como manter as interfaces por processador com um nível de complexidade aceitável, adotou-se para o processador duas portas de acesso para leitura e uma para escrita.

Esta restrição a nível físico não significa que um número maior de portas de acesso não possa ser constituído a nível lógico. Esta divisão de acesso a nível lógico significa a possibilidade de se ter acesso seletivo a conjuntos diferentes de dados solicitados à memória; a simulação mostrou que 4 acessos lógicos para leitura, por grupo, atendem a todas as situações, e este foi o número que foi adotado. Para o caminho de escrita à memória utilizam-se duas portas lógicas.

O próximo passo é determinar o número de unidades aritméticas. Cada unidade aritmética será composta de um somador, um multiplicador e um módulo divisão/raiz quadrada, todos capazes de executar um conjunto completo de operações tanto sobre números inteiros como sobre números de ponto flutuante padrão IEEE. Os dois primeiros são capazes de iniciar uma nova operação a cada ciclo; o último ficará bloqueado até que a operação tenha sido terminada.

A tabela 1 apresenta os resultados da simulação para várias combinações de número de unidades funcionais e de grupos. A análise desta tabela, sobretudo o desempenho relativo ao Cray X deve ser feito com algum cuidado, já que nenhum computador atinge o desempenho previsto, tanto em software como em hardware, em suas primeiras versões. Para a primeira versão do Ômicron deve-se esperar um desempenho da ordem de 50% do apresentado nesta tabela.

Considerando-se o o caso de um único grupo, conclui-se que uma unidade aritmética não é suficiente, e que com duas a média geométrica já está praticamente assintótica. O exame dessa tabela permite concluir que a utilização de quatro unidades ariméticas, divididas em dois grupos de duas, constitui a melhor solução.

Um efeito colateral importante desta decisão é que é possível utilizar um register file com apenas duas portas de escrita, e uma palavra de 64 bits, ambos resultados muito favoráveis e que permitem fácil implementação.

Em resumo, cada grupo incluirá duas unidades aritméticas, uma unidade de desvio, três unidades de endereço sob demanda, numeradas de 0 a 2, e três unidades de endereço múltiplo, numeradas de 3 a 5 . De cada grupo de três unidades de endereço, uma gera endereços para operações de escrita, e duas geram endereços para operações de leitura.

Cada um destes grupos admite, por sua vez, os geradores múltiplos como subgrupos.

7 – Extensibilidade e integrabilidade

A arquitetura APA permite a expansão dos grupos de unidades funcionais para números mais elevados, obtendo-se uma consistente melhoria de desempenho na maior parte dos casos. Os grupos foram limitados a dois no projeto Ômicron como conseqüência da limitação do número de acessos simultaneos à memória, e não por esgotamento do paralelismo utilizável pela arquitetura.

Como a taxa de transferência de informações entre grupos de unidades funcionais é baixa, surge naturalmente a possibilidade de integração de um grupo, com duas unidades aritméticas e respectivo register file em um único chip, viabilizando a utilização de um número maior de grupos. Embora o número de portas necessário ainda seja alto pelos padrões tecnológicos atuais, pode-se prever sua viabilidade em poucos anos.

8 - Bibliografia

[Alv90] Álvarez, J. P., e Campos, G. L., "Estudo da variação do desempenho com o número de grupos e unidades funcionais", relatório técnico 3 do projeto Ômicron, São Paulo, 1990.

[Col88] CollWel, R.P. at al., ."A VLIW Architecture for a Trace Scheduling Compiler", IEEE Trans. Comput. C-37 (8):967-979, August 1988

[Den89] Dehnert, J. C., Hsu, P. Y.T., Bratt, J. P., "Overlapped Loop Suport in the Cydra 5", 3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 26-38, April 1989

[Die88] Diede, T., et al. "The Titan Graphics Supercomputer Architecture", IEEE Computer 21 (9):13-30, SeptemberF 1988

[Dun90] Duncan, R. A "A survey of Parallel Computer Architectures", IEEE Computer 23 (2):5-16, February 1990

[Fis81] Fisher, J. A., "Trace Scheduling: A Technique for Global Microcode Compaction", IEEE Trans. Comput. C-30 (7):478-490, July 1981.

[Jou89] Jouppi, N. P. and Wall, D. W., "Available Instruction-level Paralelism for Superscalar and Superpipelined Machines", 3rd Int. Conference on Architectural Support for Programming Languages and Operating Systems, 272-282, April 1989.

[Mah86] McMahon, F. H. "The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range," Lawrence Livermore Nat'l Laboratory Report No. UCRL-53745, Livermore, CA, Dec. 1986.

[Lam88] Lam, M., "Software Pipelining: an Effective Scheduling Technique for VLIW Machines", Proc. of the Sigplan '88 Conference on Programming Language Design and Implementation, 318-328, 1988.

[Rau82] Rau, B. R., "Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing", Proc. of the 14th Annual Microprogramming.Workshop, 183-198, October 1982.

[Rau89] Rau, B. R., Yen, D. W. L. and Towle, R. A. "The Cydra-5 Departamental Supercomputer", IEEE Computer 22 (1):12-35, February 1989

[Ski88] Skillicorn, D. B., "A Taxonomy for Computer Architectures", IEEE Computer 21 (11):46-57, November 1988.

[Smi88] Smith, J. E. and Pleszkun, A. R., "Implementing Precise Interrupts in Pipelined Processors", IEEE Trans. Comput. C-37 (5): 562-573, May 1988.