Dimensionamento de processadores com arquitetura horizontal para exploração de micro e mesoparalelismo

Geraldo Lino de Campos *

Resumo

Arquiteturas horizontais são caracterizadas pela possibilidade de controlar independente e diretamente as diversas unidades funcionais de um processador, geralmente através de um único fluxo de controle. Assim, podem explorar apenas o paralelismo de grau mais fino, e dispor de um número elevado de unidades funcionais se torna inútil quando existem dependências de dados ou de controle.

O nível de desempenho pode ser elevado, para um mesmo total de unidades funcionais, se estas forem divididas em grupos, cada um com seu próprio conjunto de registradores e fluxo de controle independente. Um efeito colateral altamente desejavel é a redução do número de portas necessário no conjunto de registradores centrais, bem como no tamanho médio da instrução.

Apresenta-se resultados para um processador com arquitetura APA, mostrando que se pode obter um desempenho 2,4 vezes melhor em relação a uma configuração monolítica.

Abstract

Horizontal architectures can control several functional units independently, and usually have a single flow of control. As such, they exploit only the finer parallelism, and a large number of functional units is useless when the code contains control or data dependencies.

Performance can be increased, with the same total hardware, if the functional units are divided in groups, each with a private register file and with an independent flow of control. This will neither impair the performance when many functional units are required, nor affect the cycle time adversely. Very important side effects are the reduction on the number of ports in the central register file, with reduction of the instruction size as well.

Results are presented for a processor showing that improvements in the range of 2.4 can be achieved in the geometric mean of performance when executing the Lawrence Livermore Kernels.

* Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, São Paulo 01051 - BRASIL

Tel (55)(11)815-9322 ramal 3288

e-mail: RTC@BRFAPESP.BITNET

1. Introdução

A sempre crescente demanda por processadores de alta velocidade tem levado à introdução de pipelines mais sofisticados e paralelismo cada vez maior em todas as arquiteturas, desde micro até super-computadores. Estas técnicas, entretanto, encontram limitações: o tempo de relógio de um pipeline é determinado, em sua maior parte, pela tecnologia disponível e pela complexidade das operações necessárias em cada ciclo; por outro lado, a natureza dos programas limita o uso de paralelismo.

O propósito deste trabalho é investigar os limites de utilização prática do paralelismo em arquiteturas horizontais. As arquiteturas horizontais são caracterizadas pela possibilidade de controlar independente e diretamente as diversas unidades funcionais de um processador; representantes típicos são as arquiteturas com instruções muito grandes (VLIW - Very Long Instruction Word) [Fis83], e seus descendentes, como a Arquitetura Policíclica Assíncrona (APA), descrita em uma versão preliminar em [Cam90], e em sua forma definitiva em [Cam92a, Cam92b].

Os resultados apresentados têm, entretanto, um significado mais amplo. Como demonstrado em [Jou89], os processadores superescalares, superpipelined e VLIW são aproximadamente equivalentes em termos de desempenho e de exploração do paralelismo. Assim, pode-se esperar que os resultados apresentados neste trabalho sejam igualmente válidos para as máquinas superescalares e superpipelined.

Os limites no paralelismo disponível em programas usuais foram estudados por diversos autores, com resultados bastante discrepantes, como mostra a tabela seguinte, baseada em [Smi89] e completada com resultados mais recentes:

Autor	Ano	Grau de paralelismo em programas usuais
Weiss/Smith	1984	1.58
Tjaden/Flynn	1970	1.8
Sohi	1987	1.8
Acosta et al	1986	2.7
Kuck et al	1972	8
Reisman/Foster	1972	51
Nicolau/Fisher	1984	90
Smith et al	1989	~2
Jouppi/Wall	1989	~2 a ~5

Há várias razões para essas discrepâncias; a principal é que os autores adotam hipóteses diferentes para realizar cada estudo. É sempre possível obter um grau muito elevado de paralelismo; por exemplo, o seguinte trecho de código pode apresentar paralelismo de grau N, qualquer que seja N:

```
      DO 1 I = 1, N
1     X(I) = A(I)
```

Entretanto, é óbvio que não se pode obter esse grau de paralelismo apenas replicando N vezes uma unidade aritmética em uma arquitetura horizontal, devido a conflitos de acesso à memória e à imensa instrução que resultaria, mesmo ignorando-se a impossibilidade física de realização de uma máquina com um número elevadíssimo de portas de acesso à memória e ao conjunto de registradores do processador. A conclusão importante é que é fácil obter medidas irrealistas do grau de paralelismo existente em um trecho de código.

A medida significativa é a que expressa o grau de paralelismo que pode ser obtido em aplicações práticas e supondo um hardware com características razoáveis. Para arquiteturas com memória compartilhada é irrealista esperar mais do que alguns acessos à memória em cada ciclo, e este fator deve ser levado em conta para a obtenção de resultados significativos. Outro aspecto fundamental é considerar a latência dos acessos, uma vez que os processadores são geralmente "superpipelined", no sentido definido em [Jou89].

Destas considerações resulta que o propósito deste trabalho não é medir o paralelismo eventualmente presente no código, mas medir o aumento real de desempenho que se pode obter mediante o acréscimo de unidades funcionais de qualquer tipo (unidades aritméticas, vias de acesso à memória, etc) a uma configuração em estudo, levando-se em conta todas as restrições arquitetônicas existentes. Se o aumento de desempenho for pequeno, pode-se considerar que o grau útil de paralelismo já foi atingido.

Neste artigo, define-se paralelismo de granulação fina, ou microparalelismo, como o paralelismo que pode ser obtido dentro de um mesmo fluxo de controle, e paralelismo de granulação média, ou mesoparalelismo, como o que pode ser obtido pela execução replicada de um mesmo trecho de código. O paralelismo que pode ser obtido pela execução independente de trechos diferentes de código não será considerado neste texto.

Como este estudo foi realizado visando dimensionar o número e o tipo de unidades funcionais para a implementação de um processador APA orientado para problemas numericamente intensivos, o exemplo de programas adotado foi o conjunto dos núcleos Lawrence Livermore, bem conhecidos e considerados representativos para este tipo de processamento.

Este trabalho está organizado em 5 seções. Apresentam-se a seguir as principais características da APA. Na seção 3 apresenta-se a metodologia utilizada para conduzir o estudo de disponibilidade de paralelismo, e resultados para máquinas com fluxo de controle único. A seção 4 apresenta os resultados para múltiplos fluxos de controle; a seção 5, as conclusões.

2. Arquitetura Policíclica Assíncrona

A arquitetura Policíclica Assíncrona resultou de uma análise crítica das características da arquitetura VLIW.

Um processador VLIW é caracterizado por [Fis83]:

1 - um único fluxo de execução;

2 - um número elevado de vias de dados e de unidades funcionais, com controle planejado em tempo de compilação;

3 - instruções com um número suficiente de bits para controlar direta e independentemente, em cada ciclo, todas as unidades funcionais;

4 - operações que requeiram um número pequeno e predizível de ciclos para sua execução;

5 - cada unidade funcional deve ter a capacidade de iniciar uma nova operação em cada ciclo (isto é, devem ser pipelined).

Em termos de hardware, um processador VLIW ideal deve possuir muitas unidades funcionais ligadas a um conjunto de registradores centrais (CRC). Idealmente, cada unidade funcional deveria ter várias portas de leitura (por exemplo, duas para as unidades aritméticas) e uma porta de escrita no CRC. O CRC deve apresentar banda passante suficiente para permitir qualquer combinação de acessos gerados pelas unidades funcionais.

Infelizmente, o hardware descrito acima é irrealista; todas as realizações de arquiteturas VLIW adotaram vários compromissos para se tornarem implementáveis.

Examinando a caracterização conceitual apresentada acima, verifica-se que o propósito é definir uma arquitetura que apresente ao mesmo tempo um elevado grau de paralelismo e um ciclo de controle tão simples quanto possível, e portanto potencialmente muito rápido.

Um exame mais acurado mostra que nem todas as condições acima são necessárias. Em particular, a condição 1 não contribui para nenhum dos objetivos. É possível, pelo menos no nível conceitual, imaginar uma máquina em que cada unidade funcional tenha sua própria seqüência de instruções, seu próprio cache e lógica de controle. Apesar desta situação implicar num aumento do hardware necessário, não implica necessariamente em aumento na duração do ciclo de controle.

Outro aspecto importante é que a condição 4 não pode ser satisfeita pelas unidades funcionais responsáveis pelo acesso à memória no caso de processadores de alto desempenho, devido à necessidade de se dispor de um subsistema de memória dividido em um elevado número de bancos, o que leva a conflitos de acesso que não podem ser previstos em tempo de compilação. Este problema pode ser contornado considerando-se que o processador continue operando de forma sincronizada, porém em tempo virtual: se algum acesso ainda não estiver completo no momento esperado, o relógio do processador pára até que o acesso se complete. Esta estratégia, entretanto, pode ter um efeito extremamento adverso nos processadores de alto desempenho, uma vez que a paralização do relógio causará também a paralização das unidades encarregadas de gerar novos endereços de acesso, com efeito cumulativo.

A primeira característica distintiva da APA visa evitar este problema, o que é conseguido através do desacoplamento do processo de acesso à memória, que fica distribuído em dois tipos de unidades funcionais. O primeiro, chamado de unidade de endereço, gera e envia os endereços necessários ao subsistema de memória, que os atende individualmente, sem considerar a ordem de requisição ou outros fatores; o segundo, chamado de unidade de referência de dados, é responsável pelo reordenamento dos dados vindos da memória, e pelo seu fornecimento, em ordem, às outras unidades funcionais.

As unidades de endereço operam em dois modos: modo de endereço único e modo múltiplo. No modo de endereço único, sua função é apenas receber um endereço calculado em outras unidades funcionais e encaminhá-lo ao subsistema de memória. No modo múltiplo, sua função é a de gerar autonomamente um número determinado de endereços em um conjuto de progressões aritméticas; este modo é utilizado para referência a um conjuto de matrizes. Uma vez inicializada, uma unidade de endereços, operando em modo múltiplo, opera de maneira autônoma, independente do fluxo de controle principal, gerando novos endereços enquanto o subsistema de memória possa absorvê-los.

As conseqüências são que o subsistema de memória opera em sua capacidade máxima, o fluxo de controle principal não é afetado por uma eventual saturação do subsistema de memória (que paralisará somente a unidade de endereços) e o processo de geração de endereços também não é afetado por uma eventual paralisação do fluxo de controle principal. Sob o ponto de vista do hardware, ocorrem dois efeitos colaterais altamente desejáveis: por um lado, reduz-se o número de portas necessárias no CRC, uma vez que as unidades de endereço operam sobre um conjunto de registradores próprio, e por outro, reduz-se o número de bits necessários na instrução, já que as unidades de endereço têm operação independente .

Extendendo-se esta idéia de operação assíncrona, pode-se conceber uma arquitetura onde subconjutos de unidades funcionais podem ser separados do fluxo de controle principal, e passar a operar de maneira autônoma até a execução de uma instrução especial que os devolva ao fluxo de controle principal.

Tentando-se programar esta máquina verifica-se que esta abordagem é mais complexa que o necessário, uma vez que a divisão de unidades funcionais se dá de uma maneira bastante sistemática. Assim,

uma divisão hierárquica em dois níveis revelou-se adequada a praticamente todas as situações. Decidiu-se então permitir a divisão das unidades funcionais em *grupos*, compostos por uma unidade de controle, um certo número de unidades aritméticas e de referência de dados, com capacidade para iniciar a operação autônoma de unidades de endereço.

Verifica-se também que a comunicação entre grupos é pouco freqüente. Resulta então uma outra característica muito importante: cada grupo pode ter seu próprio CRC, o que permite a redução do número de portas a valores realizáveis. Obviamente, é necessário dispor de um mecanismo de comunicação entre grupos. Como cada grupo possui instruções independentes, somente é necessário prover instruções para os grupos que estejam ativos em um certo instante, o tamanho da instrução torna-se dinâmico sem complexidades adicionais.

O uso eficiente destas características exige que o processador apresente outras duas caracterísiticas: uso de execução antecipada e de interrupções retardadas.

Para manter as unidades funcionais ocupadas, valores devem ser calculados assim que os respectivos operandos estejam disponíveis, mesmo sem a garantia de que o fluxo de controle implicará no uso do resultado. Como exemplo, considere-se o segmento de código abaixo:

```
A = ...
B = ...
......
IF (CONDIÇÃO) A = A / B
```

A divisão, uma operação com latência elevada, deve ser iniciada assim que os valores de A e B estejam disponíveis, mesmo sem que se saiba do resultado da condição a ser testada. Se a condição for verdadeira, o resultado será utilizado; se falsa, o resultado será simplesmente desprezado.

Esta estratégia, embora eficiente, apresenta um perigo, que é o de produzir interrupções desnecessárias. Imagine-se, por exemplo, que a condição no exemplo acima fosse B.NE.0; o papel do IF seria justamente evitar a interrupção de divisão por 0. Assim, é necessário evitar a ocorrência de interrupções quando uma situação anormal é detectada. Nestes casos, o endereço da instrução ofensora é tomado como o resultado da operação, e um descritor de resultado, que existe para todas as palavras, passa a conter um código que indica qual o erro ocorrido. Interrupções só ocorrem mediante a execução de instruções determinadas, que testam o descritor de resultado, quando, pela unificação dos fluxos de controle, pode-se ter certeza de que resultados são realmente significativos. Um efeito colateral importante desta estratégia é que interrupções só podem ocorrer em situações bem determinadas, o que simplifica substancialmente o mecanismo de controle, permitindo um ciclo mais rápido.

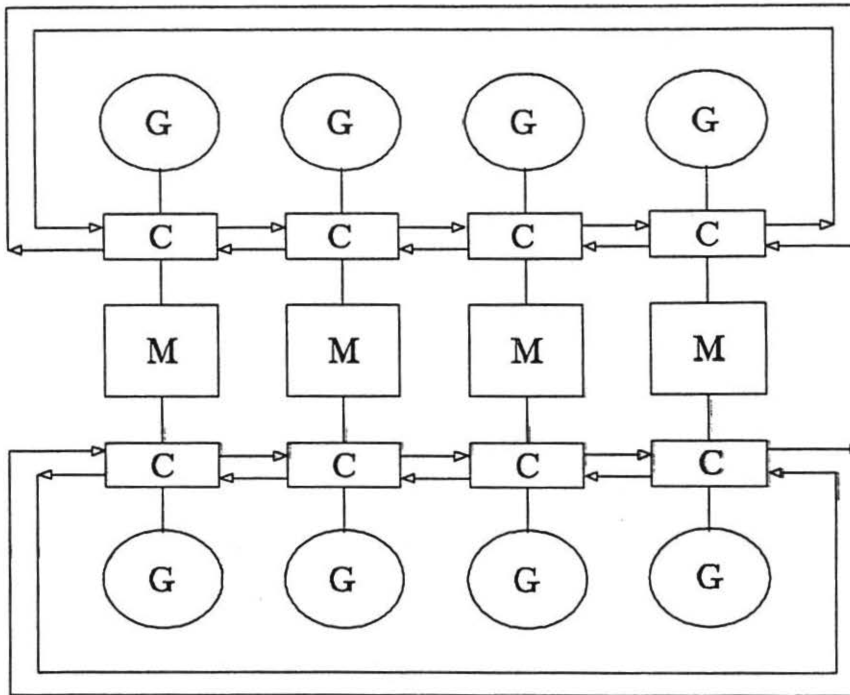


Figura 1 - 8 grupos interligados por uma rede ômicron

O descritor de resultado possui também o código de condição associado à geração do valor (se for válido), o que permite seu uso como predicado para controlar a execução de outras instruções.

A execução eficiente de loops é obtida pelo uso de uma variante do suporte policíclico; como é um mecanismo semelhante ao descrito em [Den89], não será detalhado aqui. Suas principais características são a execução automática de prólogos e epílogos e a existência de instruções com execução controlada por predicados explícitos.

O resultado da utilização destas características é um processador com desempenho bastante superior a um VLIW equivalente, exigindo CRCs com um número reduzido e exeqüível de portas, e com istrução de tamanho dinâmico. Como cada grupo é relativamente pequeno, pode ser executado sem grandes problemas de tempo de propagação, e eventualmente integrado em um único chip. Como exemplo, na máquina resultante dos estudos de dimensionamento apresentado nas seções seguintes, o CRC apresenta apenas quatro portas de leitura e duas de escrita, e a instrução ocupa apenas uma palavra (80 bits).

A figura 1 apresenta a configuração resultante: um processador APA é caracterizado pela existência de um certo número de grupos, conforme definido acima e representados pelos blocos marcados

com letra G, ligados a um subsistema de memória com 4 submódulos (identificados pela letra M) através de um mecanismo de interconexão adequado. Foi desenvolvido um mecanismo específico, chamado *chave ômicron*, que otimiza a banda passante em prejuízo do tempo de acesso; sua descrição foge ao escôpo deste trabalho e será publicada oportunamente. Estes blocos estão indicados na figura pela letra C.

Além disto, a divisão em grupos permite a exploração do mesoparalelismo mesmo nos casos em que isto não é possível num processador VLIW convencional. Isto pode ser obtido pela divisão de loops nos casos em que dependências de dados ou de controle implicam no uso de um número reduzido de unidades funcionais em cada fluxo de controle, como se verá a seguir. No texto que segue, todo o código integrante de um loop mais interno é chamado de *bloco*; não é necessariamente um bloco básico, uma vez que pode conter comandos condicionais. Um bloco pertence a uma das três categorias abaixo.

2.1 Blocos sem dependencias de dados ou de controle

Estes blocos podem ser executados inteiramente por um grupo, ou, se a relação entre operações e operandos o indicar, pode ser dividido por um número arbitrário de grupos, de modo a permitir o uso de toda a banda passante de memória e de toda a capacidade processamento dos grupos disponíveis. Esta categoria também inclui alguns casos particulares muito freqüentes na prática, nos quais a dependência de dados é facilmente contornável, como é o caso da soma de produtos apresentada no exemplo abaixo.

```
      DO 1 I = 1, N
1     S = S + A(I) * B(I)
```

Este loop pode ser dividido em

```
      DO 1 I = 1, N, 2                    DO 1 I = 2, N, 2
1     S = S + A(I)*B(I)             1     S = S + A(I)*B(I)
```

Cada um deles pode ser executado em um grupo diferente, seguido pela soma dos totais parciais. Como cada grupo possui seu próprio conjunto de registradores, não existe problema de conflito com as variáveis S e I.

2.2 Blocos com dependência de controle

Blocos com dependência de controle têm seu desempenho limitado pela latência das instruções que estabelecem a dependência e das instruções de desvio. Nos casos em que é aplicável, a execução controlada por predicado pode evitar esta latência. Uma solução mais geral, quando não existem dependências de dados concomitantes, é dividir o loop entre os grupos disponíveis, até ser atingida a banda passante de memória.

2.3 Blocos com dependência de de dados

No caso geral, é difícil ou impossível paralelizar blocos com dependências de dados. Em alguns casos triviais, como apresentado em 2.1, a dependência pode ser levantada. Nos casos em que a dependência é imediata e o bloco é pequeno, pode ser utilizada uma técnica que consiste em aumentar a distância de dependência e executar o bloco resultante em vários grupos. Esta técnica é geralmente limitada ao caso da expansão de dependência de distância 1 para distância 2, como no seguinte exemplo:

```
      DO 11 K = 2, N
11    X(K) = X(K-1) + Y(K)
```

que pode ser divido em

```
                                        X(2) = X(1) + Y(1)
      DO 11 K = 3, N, 2                 DO 11 K = 4, N, 2
11    X(K) = X(K-2) + Y(K-1) +Y(K)   11 X(K) = X(K-2) + Y(K-1) +Y(K)
```

Os casos gerais devem ser tratados com as técnicas usuais de paralelização, que não são consideradas neste trabalho; nesses casos, a APA não apresenta vantagens substanciais em relação a outras arquiteturas.

3. Metodologia e resultados para utilização de microparalelismo

Esta seção apresenta a metodologia e resultados de utilização de microparalelismo (paralelismo em um mesmo grupo), para aplicações numericamente intensivas.

Tabela 1 - Desempenho com exploração de microparalelismo

Número de unidades		Desempenho assintótico em MFlops					
aritméticas	de referência a dados*	Mínimo	Média Harmônica	Média Geométrica	Média Aritmética	Máximo	Variação da média geométrica em relação ao valor anterior (em %)
1	2 e 1	16	42	48	49	97	
2	2 e 1	20	61	84	98	187	75
3	2 e 1	20	63	93	117	273	11
4	2 e 1	20	64	98	127	348	5
5	4 e 2	20	66	104	139	401	6
6	4 e 2	20	67	115	171	528	6
7	4 e 2	20	68	116	174	528	< 1
8	4 e 2	20	68	117	182	673	< 1
...							
16	4 e 2	20	68	118	182	673	< 1 (relativo a 8)

* O primeiro número indica as unidades para leitura, e o segundo para escrita.

Para avaliação do desempenho foram utilizados on núcleos Lawrence Livermore [Mah86]; foram utilizados 22 núcleos, tendo sido removidos os de número 14 e 22, para evitar considerações de precisão nas funções transcedentais envolvidas. Como é usual nestes casos, apresentam-se os valores mínimo, máximo, e as médias harmônica, aritmética e geométrica dos resultados, considerando-se a geométrica como a mais significativa, por se aproximar mais do desempenho real.

Os núcleos foram compilados manualmente, tomando-se especial cuidado para realizar apenas as otimizações mecânicas que se pode esperar de um compilador convencional. Os efeitos de conflitos de acesso à memória foram tomados por seus valores médios. Supõe-se que o código já estivesse no cache na entrada do loop.

As hipóteses adotadas para o hardware foram baseadas em uma implementação com tecnologia ECL, utilizando componentes comercialmente disponíveis. O tempo de ciclo adotado foi de 10ns, com todas as operações em pipeline, tendo latência de 5 ciclos, com registradores de passagem operando com dois ciclos, exceto para operações de divisão e raiz quadrada, estas com latência de 25 e 45 ciclos, sem operação em pipeline.

Tabela 2 - Desempenho com micro e mesoparalelismo

Número de grupos com 2 unidades aritméticas	Desempenho assintótico em MFlops					Aumento na média geométrica em % *
	Mínimo	Média Harmônica	Média Geométrica	Média Aritmética	Máximo	
2	20	74	113	142	356	35
4	20	90	179	265	713	53
8	20	100	288	516	1426	144

* em relação a uma configuração com número equivalente de unidades aritméticas

Para uma análise inicial, considerou-se um grupo com vários números de unidades funcionais, variando-se de uma unidade aritmética e uma unidade de referência à memória até oito unidades aritméticas com 6 unidades de referência à memória - quatro para leitura, duas para escrita e número compatível de unidades de endereço.

A tabela 1 apresenta os resultados. Esta tabela mostra que existe um aumento significativo no desempenho na passagem de uma para duas unidades aritméticas, e pequenos aumentos a seguir, excetuando-se alguns núcleos muito favorávoráveis. Este resultado não é o desejado, mas apenas confirma outros resultados recentes, que indicam que o grau de microparalelismo disponível em códigos usuais é da ordem de 2.

Aliás, foi exatamente esta constatação que forneceu a idéia inicial de divisão das unidades funcionais em grupos, para permitir o aproveitamento do mesoparalelismo.

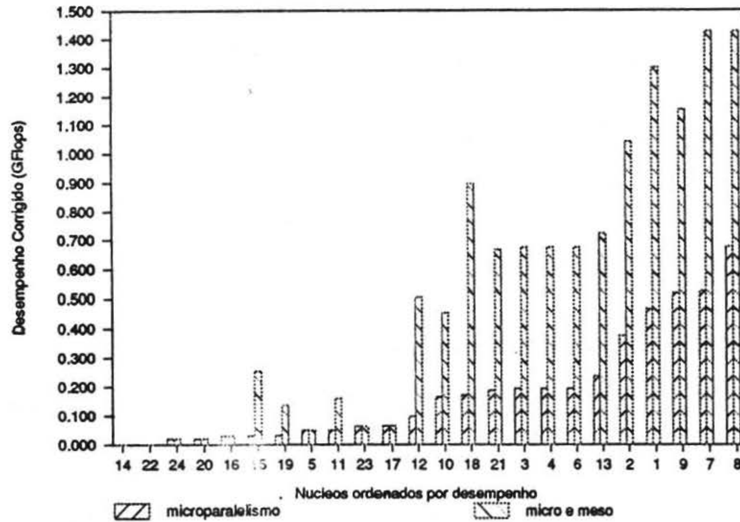
4.Resultados para utilização de mesoparalelismo

Para a realização deste estudo, adotaram-se grupos com duas unidades aritméticas e duas unidades de referência à memória, uma para leitura e outra para escrita, conforme a conclusão acima.

A Tabela 2 mostra os mesmos resultados, agora para configurações de 2 a 8 grupos, em configuração compatível com a apresentada na figura 1. Os conflitos de acesso à memória foram adequadamente considerados, supondo-se a utilização de uma chave ômicron para a interligação entre os grupos e o subsistema de memória.

Estas configurações exploram micro e mesoparalelismo, e pode-se verificar aumentos de até 144% em relação às configurações com número comparável de unidades funcionais na tabela 1. Conforme

Gráfico 1 - Desempenho com 16 unidades aritméticas



esperado, verifica-se que o aumento de desempenho é maior para configurações com mais recursos, que agora podem ser utilizados mais vezes, enquanto permaneciam ociosos na configuração anterior.

O Gráfico 1 compara o desempenho individual de cada núcleo para a situação com 16 unidades aritméticas, utilizando apenas o microparalelismo, e utilizando micro e mesoparalelismo. Este gráfico apresenta os núcleos em ordem crescente de desempenho.

Um detalhe importante é que os núcleos de desempenho médio estão incluídos entre os núcleos que apresentam maior aumento de desempenho. Como a maioria das aplicações práticas se encontra nesta faixa, conclui-se que esta melhoria ocorre nos casos em que é mais útil.

5. Conclusões

O estudo dos resultados da simulação da execução dos núcleos Lawrence Livermore mostra que a exploração do microparalelismo é pouco útil em configurações com mais de duas unidades aritméticas de ponto flutuante; para arquiteturas convencionais, esta conclusão implica na existência de duas outras unidades aritméticas, que podem ser de ponto fixo, para o cálculo de endereços.

O estudo destes mesmos resultados mostra que a exploração do mesoparalelismo permite elevar substancialmente o desempenho, justificando-se a utilização de até 8 grupos com a configuração descrita acima.

A conclusão final é que a divisão das unidades funcionais de um processador VLIW em grupos de unidades funcionais autônomas não compromete o desempenho da máquina original nos casos em que se consiga utilizar inteiramente suas unidades funcionais, e que permite um aumento substancial de desempenho nos casos em que se possa utilizar o mesoparalelismo. Esta divisão permite ainda simplificar a implementação, uma vez que necessita de um conjuto de registradores centrais com um número muito menor de portas.

Bibliografia

[Cam90] Campos, G. L., "Arquitetura Policíclica Assíncrona", Anais do III Simp. Brasileiro de Arquitetura de Computadores, 83-95, Rio de Janeiro, novembro de 1990

[Cam92a] Campos, G. L., "Asynchronous Polycyclic Architecture: an overview", Proc. of the 12th Word Computer Congress, vol I- Algorithms, Software, Architecture, Madrid, Sept 1992.

[cam92b] Campos, G. L., "Asynchronous Polycyclic Architecture", Proc. of the 6th International Conference on Parallel Processing", Lyon, Sept 1992.

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

[Fis83] Fisher, J. A. "Very Long Instruction Word Architectures and the ELI-512", IEEE Conf. Proc. of the 10th Annual Int. Symp. on Comput. Architecture, 140-150, June 1983.

[Jou89] Jouppi, N. P. and Wall, D. W., "Available Instruction-level Parallelism 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.

[Smi89] Smith, M. D., Johnson, M. and Horowitz, M. A. "Limits on Multiple Instruction Issue", 3rd Int. Conference on Architectural Support for Programming Languages and Operating Systems, 290-302, April 1989.