Implementação Paralela do Filtro de Kalman para Aplicações em Tempo Real

RESUMO

Neste trabalho apresenta-se o desenvolvimento e implementação paralela do filtro Kalman. O processamento numérico é efetuado em uma arquitetura MIMD baseada em processadores digitais de sinal de terceira geração com ponto flutuante, *pipelining* e arquitetura Harvard. Aspectos de partição do algoritmo, *scheduling* de processadores e *deadlock* são abordados. Um método heurístico para paralelização de algoritmos seriais é proposto. O *speedup* é obtido implementando-se, em tempo real, o filtro de Kalman nas versões serial e paralela, e aplicando-o para estimar os estados de um sistema dinâmico com ruído de medidas.

ABSTRACT

This paper describes the implementation and performance evaluation of a parallel Kalman filter. The parallelism in the Kalman filter equations are exploited by means of a suitable task partition and scheduling among processors in a DSP-based MIMD architecture. Due to its high floating point number crunching capability and flexibility, this architecture is quite attractive for real time applications requiring numerically intensive algorithms. Details concerning interprocessor communication and deadlocks are discussed. A general heuristic approach for paralellization of serial algorithms is also proposed. A real time application of the Kalman filter is then presented, aimimg at assessing the speed up obtained with the parallel implementation.

WALTER ABRAHÃO DOS SANTOS
INPE-LIT-VIT - CP 515, 12970-S.J. DOS CAMPOS-SP
E-mail: WASKY@LIT.INPE.BR

ELDER MOREIRA HEMERLY
CTA-ITA-IEEE
12228-900-S.J. DOS CAMPOS-SP

1. Introdução

Algoritmos sofisticados de filtragem digital são, em geral, computacionalmente exigentes. Em várias aplicações em tempo real tem-se adicionalmente a necessidade de altas taxas de amostragem. Estes fatores usualmente inviabilizam o emprego de sistemas monoprocessadores em tais aplicações. Com o advento de tecnologia VLSI e desenvolvimento de processamento paralelo, estas aplicações podem ser viabilizadas utilizando-se sistemas multiprocessadores, de baixo custo e alto desempenho. Este artigo apresenta uma aplicação típica, relativa à implementação do filtro de Kalman em uma arquitetura MIMD baseada em processadores digital de sinais(DSP's) tipo TMS320C30, da baixo custo e alto desempenho.

Inicialmente considera-se o problema da implementação de filtros digitais, com enfoque especial para aplicações do filtro de Kalman em tempo real. Em seguida descreve-se a arquitetura VLSI paralela empregada para o processamento do algoritmo e aquisição das medidas. O procedimento utilizado para paralelizar o filtro de Kalman pode ser eficientemente estendido para a paralelização de outros algoritmos seriais, objetivando implementação em multiprocessadores homogêneos, podendo ser adaptada para casos mais gerais com multiprocessadores heterogêneos. Finalmente, descreve-se a aplicação em tempo real do filtro de Kalman para estimar os estados de um sistema dinâmico, efetuando-se também o cálculo dos tempos de processamento envolvidos nas implementações serial e paralela.

2. Filtragem Digital de Sinais

Apesar da implementação de filtros digitais tornar-se muito atrativa com os avanços na micro-eletrônica, há alguns problemas típicos:

(1) a quantização do sinal e coeficientes, especialmente em processadores de ponto fixo;
(2) a maioria dos algoritmos empregados na prática possuem forma serial;
(3) a baixa velocidade de processamento torna-se um obstáculo em aplicações onde tempo de execução é um fator crítico e

(4) a ausência, em certos casos, de ferramentas de software e/ou linguagens de alto nível para geração de códigos.

O filtro de Kalman é um algoritmo recursivo que permite estimar, de modo ótimo se certas hipóteses forem satisfeitas [1], as variáveis de estado de um sistema dinâmico sujeito a ruídos de estado e de medida. Devido à sua recursividade e otimalidade, este filtro tem sido amplamente utilizado em diversas áreas [2]. Por outro lado, sua implementação em tempo real requer a consideração dos seguintes fatores:

(1) ciclo de instrução (leitura, escrita, adição, multiplicação);
(2) tamanho e forma de acesso à memória;
(3) comprimento da palavra e formas de arredondamento;
(4) quantização efetuada durante leitura via A/D;
(5) conjunto de instruções, e
(6) capacidade de cálculo e tipo de aritmética.

Algumas aplicações, tais como rastreio de alvos via radar ou sonar, controle adaptativo e sistemas tolerantes a falha, exigem filtragem em tempo real. Nestes casos a aplicabilidade do filtro de Kalman fica limitada devido às operações matemáticas relativamente complexas envolvidas no algoritmo, tais como adição, multiplicação e inversão de matrizes. Dentre estas operações, a inversão matricial é a mais difícil de ser implementada em termos de velocidade de execução e precisão.

Neste trabalho mostra-se que o filtro de Kalman pode ser utilizado em aplicações que requerem altas taxas de amostragem. Para tanto emprega-se uma arquitetura com dois DSP's e propõe-se um procedimento para se paralelizar o algoritmo básico do filtro de Kalman. Apresenta-se também uma aplicação em tempo real, avaliando-se os tempos de processamento da implementações serial e paralela.

3. Arquitetura MIMD para Implementação Paralela do Filtro de Kalman

A arquitetura empregada neste trabalho é descrita com detalhes em [3], sendo uma versão preliminar encontrada em [4]. O núcleo da estrutura da arquitetura, mostrada na Fig.1, compõe-se dos seguintes processadores:

(1) Microcomputador hospedeiro (geração de código, interface amigável com usuário, gerenciamento da arquitetura).

(2) Microcontrolador PCB80C552 (aquisição de dados, interface com a planta, execução de tarefas de baixo processamento, comunicação com microcomputador hospedeiro). Na Fig. 1 mostra-se a interface com o sistema dinâmico de segunda ordem considerado na seção V.

(3) Dois processadores digitais de sinais em ponto flutuante TMS320C30 (processamento numérico intensivo, comunicação com microcomputador hospedeiro e passagem de *status*).

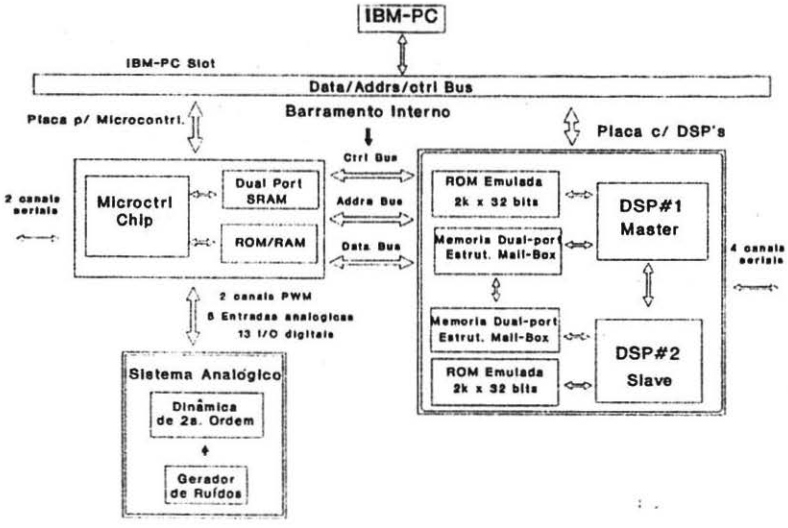

Fig. 1 - Arquitetura paralela para implementação do filtro de Kalman em tempo real.

A comunicação entre processadores ocorre através de memória

compartilhada e interrupção. A proteção de dados é efetuada particionando-se o espaço endereçável de cada processador em faixas convenientes. Os conflitos de acesso são solucionados priorizando-os de forma tal que os DSP's atuem como mestre.

A arquitetura é flexível e modular o suficiente para ser expandida e suportar a interligação de mais placas clones em hipercubo. Entretanto, à medida que o sistema se expande, deve-se estar atento às restrições de custo e contenção de recursos compartilhados.

4. Paralelização do Algoritmo do Filtro de Kalman

Nesta seção é apresentada a formulação original do filtro de Kalman, e a seguir explora-se a concorrência natural das equações. Realiza-se então uma partição de *tasks* e *scheduling* para uma arquitetura com dois DSP's trabalhando em um esquema produtor-consumidor fortemente acoplado. Como o objetivo básico é ilustrar um método de paralelização, não se efetuou partições que implicassem baixa granulariedade, e o *scheduling* obtido não é ótimo em relação ao menor tempo de execução.

Um resumo das equações do Filtro de Kalman é apresentada a seguir. Para maiores detalhes, vide [6]. Considere um sistema dinâmico descrito pela equação de estado

$$x(k+1) = A(k)\,x(k) + B(k)u(k) + G(k)\,w(k) \qquad (4.1)$$

e pela equação de medida

$$y(k) = C(k)\,x(k) + F(k)\,v(k) \qquad (4.2)$$

onde $x(k)$ é o vetor de estados n-dimensional, $A(k)$ é a matriz nxn da dinâmica do sistema, $u(k)$ é vetor de controle r-dimensional, $B(k)$ é a matriz de controle rxn , $C(k)$ é a matriz mxn de medidas, $y(k)$ é o vetor de medidas m-dimensional, $G(k)$ é a matriz nxp intensidade do vetor ruído de estado p-dimensional $w(k)$, e $F(k)$ é a matriz mxr intensidade do vetor ruído de medida r-dimensional $v(k)$. Os vetores de ruído $w(k)$ e $v(k)$ são supostos terem as

seguintes propriedades, onde $\delta(k\text{-}j)=1$, para $j=k$, e zero caso contrário,

$$E[\, w(k)\,] = 0, \qquad E[\, w(k)\, w^T(j)\,] = P_w(k)\,\delta(k-j) \tag{4.3}$$

$$E[\, x(k)\, w^T(k)\,] = 0, \qquad E[\, v(k)\, v^T(j)\,] = P_v(k)\,\delta(k-j) \tag{4.4}$$

$$E[\, x(k)\, v^T(k)\,] = 0, \qquad E[\, w(k)\, v^T(k)\,] = 0 \tag{4.5}$$

Com a hipótese de que as matrizes $A(k)$, $C(k)$, $P_w(k)$ e $P_v(k)$ são conhecidas, e o controle $u(k)$ é determinístico, o filtro de Kalman é descrito pelas equações, para $k=1,2,\ldots$,

$$\hat{x}(k+1) = A(k)\,\hat{x}(k) + B(k)u(k) + K(k)\Big(y(k)-C(k)\hat{x}(k)\Big) \tag{4.6}$$

$$K(k) = A(k)P(k)C^T(k)\Big(C(k)P(k)C^T+F(k)\,P_v(k)\,F^T(k)\Big)^{-1} \tag{4.7}$$

$$P(k+1) = A(k)P(k)A^T(k) - A(k)P(k)C^T(k)\Big(C(k)P(k)C^T(k)+F(k)P_v(k)F^T(k)\Big)^{-1}\cdot C(k)P(k)A^T(k) + G(k)P_w(k)G(k) \tag{4.8}$$

com condições iniciais

$$\hat{x}(0) = 0\ , \quad P(0) = cI_{n\times n},\ c\in R^+ \tag{4.9}$$

O vetor $\hat{x}(k)$ representa a estimativa ótima de $x(k)$ com base na seqüência de medidas $\{y(1), y(2), \ldots, y(k\text{-}1)\}$. A matriz $P(k+1)$ corresponde à covariância do erro de estimação $\tilde{x}(k+1)=x(k+1)-\hat{x}(k+1)$.

Para avaliação de *speedup* fornecido pela implementação paralela e auxílio na formulação do problema de paralelização, será tomado como ponto de partida a implementação serial do filtro de Kalman escrita em C ANSI, mostrada na Fig.2.

```
                    /* FILTRO de KALMAN SERIAL */
main()
{
void apctrans(); /* Calcula AP(k)C^T. */
void cpctrans(); /* Calcula (CP(k)C^T+FPvF^T)^-1. */
void apatrans(); /* Calcula AP(k)A^T. */
void download(); /* Passa x̂(k) para host */
void c_int02(); /* Leitura de y(k) */
/* Inicialização de matrizes e parâmetros */
{ ... }
while(1<3) /* Loop infinito */
{
        asm(" IDLE"); /* Aguarda interrupção para leitura de y(k) */
        /* Passagem de x̂(k) para host */
        download();
        apctrans(); /* Calcula apct[][]=AP(k)C^T. */
        cpctrans(); /* Calcula cpctinv=(CP(k)C^T+FPvF^T)^-1. */
        /* Cálculo de K(k) = apct[][]*cpctinv */
        { ... }
        /* Cálculo de x̂(k+1). */
        { ... }
        apatrans(); /* Calculate apat[][]=AP(k)A^T. */
        /* P(k+1)=AP(k)A^T-(AP(k)C^T).(CP(k)C^T+FPvF^T)^-1.(AP(k)C^T)^T+ GPwG^T */
        { ... }
}
```

Fig. 2 - Código serial em C ANSI do algoritmo do filtro de Kalman.

O processamento pode ser particionado de forma a dividir o número de operações matemáticas entre os dois DSP's, para detalhes teóricos vide [4]. Para tanto são definidas convenientemente as *tasks* (Tabela 1), aqui consideradas procedimentos indivisíveis e, portanto, processadas apenas por um DSP.

Após o particionamento, são explicitadas as relações de precedência e concorrência (Fig. 3) entre cada *task*. *Tasks* alinhadas horizontalmente indicam precedência no sentido de que a *task* mais à esquerda deve ser processa antes da *task* mais à direita. *Tasks* alinhadas verticalmente indicam concorrência no sentido de que todas podem ser efetuadas simultaneamente, entretanto *tasks* mais ao topo de uma mesma coluna têm maior premência de serem processadas. Linhas unindo *tasks* simbolizam a passagem de dados que servem de entrada para *tasks* posteriores. *Tasks* inter-dependentes escalonadas para processadores diferentes trocam dados via memória de duplo acesso, o que repereserita menor probabilidade de contenção em memória, visto ser possível o acesso simultâneo por dois processadores.

Tabela 1 - Partição em *tasks* do filtro de Kalman.

No. task	Definição da *task*
T0	C_INT02()- Acesso a memória dual para leitura de medida
T1	DOWNLOAD()- Acesso a memória dual para escrita de $\hat{x}(k)$
T2	$A(k)\ P(k)\ C^T(k)$ - Cálculo apct
T3	$(C(k)P(k)C^T(k)+F(k)\ P_v(k)\ F^T(k))^{-1}$ - Cálculo de cpctinv
T4	apct[][]*cpctinv - Cálculo de $K(k)$
T5	$\hat{x}(k+1)=A(k)\hat{x}(k)+B(k)u(k)+K(k)(y(k)-C(k)\hat{x}(k))$ - Cáculo de $\hat{x}(k+1)$
T6	$A(k)P(k)A^T(k)$ - Cálculo de apat
T7	apat-apct.cpctinv.apact^T+$G(k)P_wG^T(k)$ - Cálculo $P(k+1)$

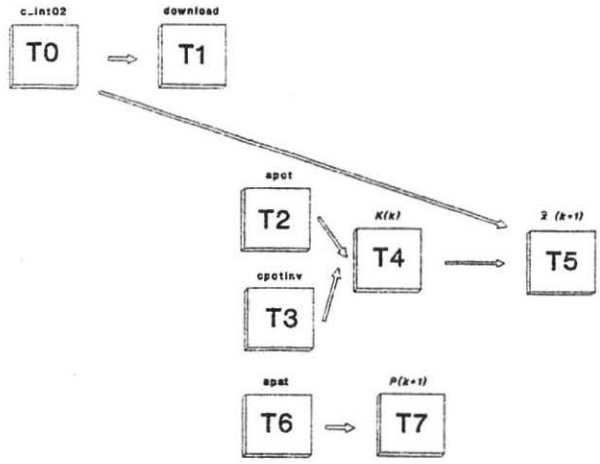

Fig. 3 - Relação de precedência e concorrência entre *tasks* no algoritmo do filtro de Kalman.

Uma vez explicitadas as relações de precedência e concorrência entre *tasks*, pode-se escaloná-las de forma a balancear a carga computacional entre os processadores (Fig. 4). Modelos probabilísticos e determinísticos podem ser utilizados para análise do problema de *scheduling*. A determinação de um algoritmo ótimo para de resolução deste problema para multiprocessadores é

um problema NP-*hard*. Contudo, alguns algoritmos dinâmicos sub-ótimos para *scheduling* têm bom desempenho [7]. Neste artigo, o procedimento de *scheduling* será tratado heuristicamente, tomando como parâmetros o tempo de execução de cada *tasks*, operações de acesso a memória e *interlock* entre processadores. O sincronismo entre processadores foi efetuado forçando-se a entrada em *IDLE* de um dos processadores, e esperando-se até que o outro processador o interrompa.

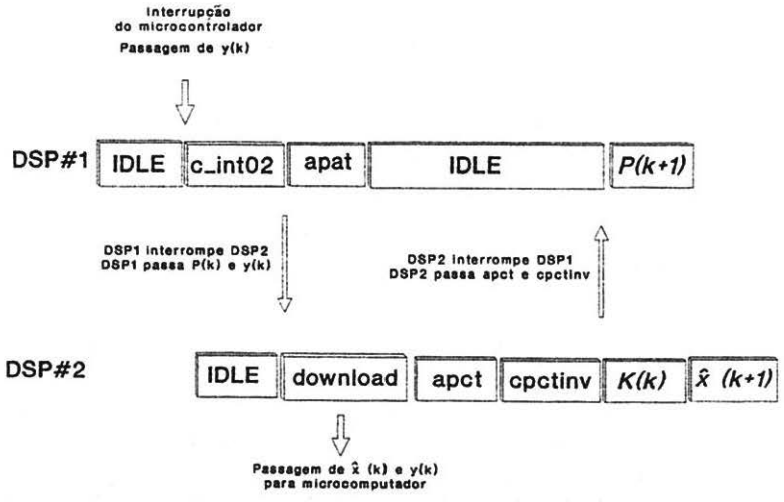

Fig. 4 - *Scheduling* de processadores.

A abordagem proposta neste trabalho para paralelização de algoritmos seriais pode ser formalizada, em linhas gerais, com os seguintes passos:

(1) Encontrar um algoritmo serial que resolva o problema de maneira eficiente e satisfaça as restrições da arquitetura-objeto.

(2) Analisar o algoritmo buscando estruturas em *loop* ou recursivas.

(3) Identificar possíveis estruturas responsáveis pelo maior parte do tempo de processamento, geralmente procedimentos repetidos a maior parte do tempo.

(4) Mediante dados do itens 2 e 3, arbitrar *tasks* dentro do algoritmo e explicitar as relações de precedência e concorrência entre elas.

(5) Quantizar o impacto de cada *task* assinalada sobre o fluxo do processamento. Para arquiteturas MIMD homogêneas, há, pelo menos, três métodos de quantização possíveis:

(5.1) Implementação do algoritmo serial escolhido diretamente no processador-objeto e, por meio de *timers* ou similares, avaliar o tempo efetivo de processamento de cada *task*.

(5.2) Simulação do sub-item anterior quer por meio de simuladores comerciais do processador-objeto, quer por meio de software escrito refletindo seus detalhes de hardware e software,

(5.3) Inferência do número de ciclos necessários para cada *task*. Obviamente, o primeiro método é o mais preciso.

(6) Escalonar convenientemente *tasks* entre processadores, e para tanto pode-se adotar uma heurística simples [8]:

(6.1) Tempos inevitáveis de inatividade devem ser distribuídos eqüitativamente entre o maior possível número de processadores.

(6.2) Em nenhum istante de tempo todos processadores podem estar simultaneamente inativos.

(6.3) *Tasks* que não possuam sucessores devem ser escalonadas por último, salvo se escalonadas em períodos de inatividade.

(7) Estabelecer os pontos de interação entre *tasks* e adicionar estruturas de sincronização entre as mesmas. Granularidade finas podem melhorar o balanceamento de carga entre processadores, mas exigirão maior precisão no sincronismo.

(8) Identificar situações potenciais de *deadlock* entre processos e eliminá-las, através de novo rearranjo de *tasks* ou sincronismo.

Uma alternativa para a automação na geração de programas paralelos é a construção de um compilador para uma linguagem de alto nível padrão, como C por exemplo. A partir do compilador, monta-se um conjunto de bibliotecas que podem ser mapeadas em multiprocessadores sem grandes problemas. Porções da biblioteca que não sejam eficientes em implementação paralela podem ser executadas serialmente. Apesar da eficiência do código gerado por compiladores ser, em geral, menor do que a gerada codificando *manualmente* programas paralelos, a vantagem no caso de compiladores é a velocidade de conversão. O código assim gerado poderá ser otimizado *manualmente* ou aplicado a outro compilador mais eficiente.

5. Aplicação do Filtro de Kalman Paralelo

Para efeito de aplicação, considerou-se o sistema

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0{,}9919 & 0{,}0608 \\ -0{,}2431 & 0{,}8724 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0{,}0081 \\ 0{,}2431 \end{bmatrix} u(k) + \begin{bmatrix} 0{,}0081 \\ 0{,}2431 \end{bmatrix} w(k) \qquad (5.1)$$

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + v(k) \qquad (5.2)$$

com P_w=0,1 e P_v=0,05. De modo a inserir a característica de tempo real ao problema, simulou-se no computador analógico COMDYNA GP6 a versão contínua do modelo (5.1)-(5.2) e utilizou-se o gerador de ruídos HP3722A, para gerar o ruído de medida $v(k)$. A leitura de $y(k)$ foi efetuada por intermédio de microcontrolador PCB80C552 com conversor A/D de 10 bits *onchip*, sendo os gráficos obtidos com um microcomputador IBM-PC. Uma vez adquirida a leitura de $y(k)$, o microconcontrolador interrompe DSP#1.

As estimativas obtidas em uma realização típica são mostradas na Fig. 5, para $u(k)$=1, $0 < k < 75$, a partir dos dados armazenados pelo DSP#2 na memória RAM de duplo acesso e transferidas ao microcomputador de interface com usuário, que gera os gráficos. No primeiro gráfico da Fig. 5 tem-se as variáveis $y(k)$ e a estimativa $\hat{x}_1(k)$, explicitando-se assim o desempenho do filtro. No segundo gráfico tem-se a evolução da estimativa $\hat{x}_2(k)$.

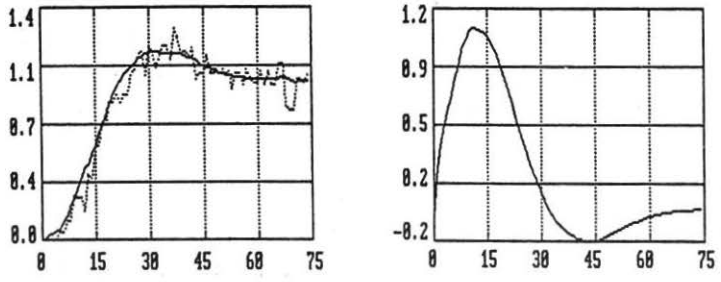

Fig. 5 - Leitura $y(k)$ e estimativa $\hat{x}_1(k)$, e estimativa $\hat{x}_2(k)$.

De modo a se determinar o tempo necessário para o processamento do filtro de Kalman no DSP TMS320C30, utilizou-se *timers* internos à arquitetura dos DSP. Desta forma foi obtido o número de ciclos necessários tanto na versão serial quanto na versão paralela do filtro de Kalman. Para a aplicação considerada, verificou-se que a implementação serial, isto é, com um DSP, requer aproximadamente 146μs, enquanto que a implementação paralela, utilizando dois DSP's, requer aproximadamente 88μs. Logo, obtém-se um *speedup* de aproximadamente 166% com a implementação paralela. Convém mencionar, para efeitos comparativos, que o algoritmo serial implementado em microcomputador IBM-PC/AT-12MHz, com coprocessador, é executado em aproximadamente 5ms.

Aumento do *speedup* pode ser obtido se elevarmos a dimensão do vetor de estados (n) ou do vetor de medidas (m). Entretanto, ganhos em *speedup* com relação às medidas são menores do que com relação aos estados. Isto se deve ao baixo paralelismo existente nas equações de medidas para uma arquitetura com dois DSP's. Este fato pode ser contornado com a adição de um terceiro DSP para processá-las separadamente.

6. Conclusões

O problema de paralelização do algoritmo serial do filtro de Kalman foi considerado neste trabalho. Um exemplo de aplicação em tempo real do algoritmo paralelo obtido foi apresentado. A arquitetura empregada para a implementação paralela compõe-se de dois DSP's para processamento numérico, microcontrolador para interface flexível com sistemas analógicos e um microcomputador para interface com usuário e plataforma de desenvolvimento de programas. Mesmo com um *scheduling* entre processadores não plenamente otimizado e com granularidade grossa obteve-se um *speedup* da ordem de 166%.

A metodologia de paralelização prosposta é geral o suficiente para permitir fácil adaptação em outros casos similares. Em vista da escassez de algoritmos paralelos e da existência de vasta documentação de algoritmos seriais eficientes para diversos problemas, a metodologia proposta torna-se ferramenta útil no mapeamento de estruturas seriais para as correspondentes

paralelas em uma determinada arquitetura. A automação destes procedimentos poderá servir de base para a montagem de um compilador que paralelize rapidamente, mesmo que sem grande eficiência, estruturas seriais básicas.

AGRADECIMENTOS:
Este trabalho foi desenvolvido com apoio financeiro da FAPESP (Fundação de Amparo à Pesquisa no Estado de São Paulo), sob processo no. 91/1009-0.

7. Referências Bibliográficas

[1] MAYBECK P.S. - *Stochastic Models, Estimation and Control - Vol. 1* - Academic Press, N. York, 1979.

[2] GELB, A. - *Applied Optimal Estimation* - The M.I.T. Press, Cambridge, 1974.

[3] STONE, H. S. - *High-Performance Computer Archtectures* - Addison-Wesley Pub. Company, Reading, 1987.

[4] SANTOS, W.A.. DOS & HEMERLY, E. M. - "Implementation anbd Performance Evaluation of DSP-based Architectures for Filtering and Control Applications" - Accepted for presentation at *International Conference on Systems Science XI*, Wroclaw, Poland, September, 1992.

[5] SANTOS, W.A.. DOS & HEMERLY, E. M. - "Arquitetura Multiprocessadora Heterogênea Aplicada ao Controle Adaptativo de Manipuladores Robóticos", *Jornada EPUSP/IEEE em Sistemas de Computação de Alto Desempenho*, São Paulo, pp. 27-34, Março, 1991.

[6] BROWN, R.G. - "Introduction to Random Signal Analysis and Kalman Filtering", John Wiley & Sons, New York, 1983.

[7] HWANG, K. & BRIGGS,F.A. - *Computer Architecture and Parallel Processing* - MacGraw-Hill, New York, 1985.

[8] LUH,J.Y.S. & LIN,C.S. - "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator", *IEEE Trans. on Syst. Man and Cybern.*, V. SMC-12, No. 2, Mar/Apr. 1982.