Paralelização em dois níveis do Modelo Regional de Previsão de Tempo – Eta

Simone Tomita, Luiz Flavio Rodrigues, Jairo Panetta
Centro de Previsão de Tempo e Estudos Climáticos, CPTEC / INPE – Cachoeira Paulista (SP)
tomita@cptec.inpe.br, lflavio@cptec.inpe.br, panetta@cptec.inpe.br

Resumo

Este trabalho apresenta a metodologia utilizada e os resultados obtidos na paralelização em dois níveis – vetorização e paralelização OpenMP – do modelo numérico de previsão do tempo Eta, para a máquina de processamento vetorial e paralelo de memória central NEC-SX6.

Demonstramos as principais vantagens do uso do padrão OpenMP, viabilizando a portabilidade de programas seriais para ambientes paralelos de memória compartilhada. A metodologia utilizada, as dificuldades de acomodar paralelismo em dois níveis e os resultados obtidos fazem parte deste trabalho.

1. Introdução

O modelo regional de previsão de tempo – Eta, foi desenvolvido na Universidade de Belgrado em cooperação com o Instituto de Hidrometeorologia da Iugoslávia e o *National Center for Environmental Prediction* – NCEP em Washington [1][2]. Em meados de 1999 a versão operacional (seqüencial) do NCEP foi instalada no CPTEC. Após testes de validação tornou-se a versão operacional, gerando a previsão diária de tempo para todo o território nacional.

Este trabalho faz parte de um projeto que visa a modernização, otimização e paralelização dos códigos de previsão de tempo do CPTEC. O projeto de modernização do código do Eta gerou uma nova versão com modificações computacionais profundas para prover paralelismo de dois níveis – vetorização e paralelismo OpenMP [3] sem comprometer a portabilidade, atendendo os objetivos do projeto. Resulta código portátil com paralelismo também portátil para máquinas de memória compartilhada.

Descreve-se a seguir, as técnicas de otimização usadas, a estratégia de paralelização aplicada e os resultados obtidos.

2. Passos Preliminares

O modelo integra, ao longo do tempo, o estado da atmosfera em área geográfica limitada, utilizando a coordenada vertical Eta [4].

O programa original foi escrito em Fortran 77, com técnicas obsoletas de programação, estrutura inadequada para paralelismo e baixo desempenho vetorial. Os principais procedimentos não tinham argumentos. Atuavam sobre dados globais ao programa, armazenados em 54 *commons*. O uso de *commons* obrigava alocação estática de variáveis e dificultava a detecção das variáveis de entrada e saída dos procedimentos. O entendimento dos fluxos de dados e de controle do programa ainda era prejudicado pelo uso de *equivalence* e *go to*.

Os objetivos da primeira fase de modificações do programa foram aumentar sua legibilidade e explicitar o fluxo de dados entre os procedimentos. Para tanto, foram necessários três passos. No primeiro passo, converteu-se o programa para Fortran 90, declarando todas as variáveis, eliminando *equivalences* e substituindo *go to* pelas estruturas de controle clássicas. No segundo passo, foram substituídos os *commons* por *modules*. Procedimentos que referenciavam variáveis em *commons* passaram a referenciá-las por *use only*. Essa prática de programação permitiu enumerar (na cláusula *only* do *use*) apenas as variáveis globais realmente utilizadas pelo procedimento. No terceiro passo, foram identificadas a intenção (*intent*) dos argumentos de todos os procedimentos, declarando-os como entrada, saída ou ambos. Simultaneamente, foram declaradas a intenção das variáveis globais a cada procedimento, por meio de comentários afixados após cada variável global presente na cláusula *only* do *use*. Como resultado, para entender o fluxo de dados de entrada e saída de cada procedimento basta observar as declarações das variáveis do procedimento, apesar do uso de variáveis globais.

Os objetivos da segunda fase de modificações foram simplificar o entendimento da árvore de chamadas dos procedimentos e reduzir a manipulação de arquivos. Para tanto, a estrutura de arquivos original (100 arquivos contendo 80 procedimentos e 54 *commons*) foi re-organizada. Criaram-se quatro *modules* temáticos - Radiação, Turbulência, Convecção e I/O. Os três primeiros tratam dos processos físicos do modelo e

contém um único procedimento invocável externamente ao módulo (os demais procedimentos são privados). Resultam 58 arquivos com 30217 linhas e a partição da árvore de chamadas em seis blocos: Inicialização, I/O, Convecção, Turbulência, Radiação e Dinâmica.

Após as duas fases de modificações já relatadas, o programa está preparado para análise de dependência de dados, vetorização e paralelização.

As grandezas meteorológicas tri-dimensionais são implementadas por *arrays* indexados por longitude(I), latitude(J), vertical(L). De forma geral, a estrutura de laços tem o seguinte formato:

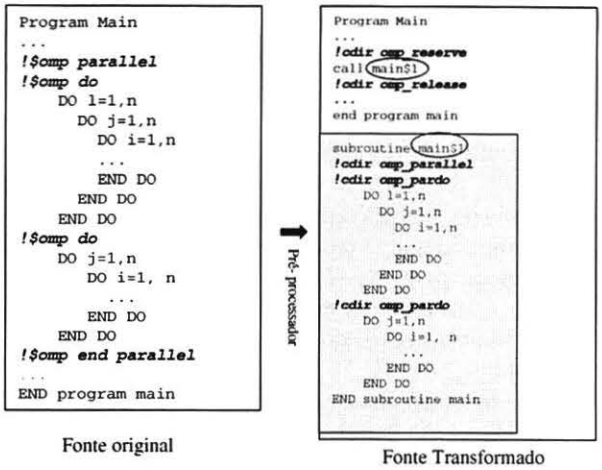
```
DO L=1, lm              → Camadas verticais
   DO J=1, jm           → Pontos na latitude
      DO I=1, im        → Pontos na longitude
         ... (bloco)
      END DO
   END DO
END DO
```

Algumas estruturas de laços invocam procedimentos ou árvores de procedimentos e algumas variações na ordem do aninhamento de laços são encontradas no decorrer do código. Como há formas bastante eficientes de paralelizar e vetorizar, um fator importante na análise é adotar critérios de decisão entre vetorização e paralelização, quando ambos são possíveis no mesmo aninhamento. Esses aspectos serão discutidos nas estratégias de paralelização.

3. Implementação do Padrão OpenMP no NEC/SX6

O ambiente de programação proposto pelo padrão OpenMP baseia-se na arquitetura de compartilhamento de memória e é considerado um padrão fácil de usar. O compilador Fortran90/OpenMP do NEC SX6 está de acordo com *OpenMP Fortran Application Program Interface Ver 1.0 - Oct 1997.*

Trata-se de um pré-processador que transforma o padrão OpenMP em diretivas privativas da NEC que são mapeadas em um conjunto de microtarefas. Tarefas são geradas quando criadas as regiões paralelas em OpenMP e são destruídas quando essas regiões são fechadas. Cada região paralela aberta é traduzida pelo compilador que separa essa porção paralela do código em um novo procedimento, como mostra a Figura 1. Esse procedimento contém as diretivas da NEC.



Fonte original — Fonte Transformado

Figura 1. Pré-processador OpenMP NEC/SX6

Para aumentar a eficiência do programa é preciso minimizar o número de regiões paralelas, maximizando o número de linhas de código de cada região. Isso se deve ao *overhead* de criação e destruição das regiões paralelas. No caso do modelo Eta, devido à complexidade do código seqüencial e de sua implementação, reduzir o número de regiões paralelas é uma tarefa difícil. A solução foi abrir e fechar regiões paralelas no escopo de cada módulo ou procedimento.

4. Estratégias de paralelização

4.1 Medidas de desempenho

O procedimento adotado para análise de desempenho foi identificar pontos críticos onde o tempo de execução comprometia o desempenho do programa. Utilizou-se *ftrace* [5], uma ferramenta proprietária da NEC para a análise detalhada da eficiência de execução de cada procedimento. Essa ferramenta ordena os procedimentos por tempo de execução, informando o número de vezes que o procedimento é invocado, a velocidade de processamento ponto-flutuante (em *Mflops*), o grau de vetorização, o tamanho médio dos vetores, etc. Tal ferramenta foi extensamente utilizada para priorizar procedimentos candidatos à otimização e avaliar os resultados. Procedimentos foram escolhidos pela combinação de alto tempo de execução com baixa complexidade da implementação.

No caso do paralelismo OpenMP, a ferramenta fornece volume excessivo de informações, pois reporta os parâmetros de execução acima citados, para cada região paralela de cada *thread*. Em primeira análise, é mais conveniente instrumentar manualmente o código, medindo o tempo de execução de cada módulo por meio de *system clock* (função intrínseca de Fortran 90). Em segunda instância, quando necessário, utilizou-se *ftrace*.

4.2 Primeiro nível de paralelização

A primeira etapa da paralelização visava aumentar a vetorização. A vetorização é entendida como o primeiro nível de paralelismo devido ao uso dos *pipelines*. É chamada de execução paralela por *hardware*.

Nesta etapa, utilizou-se técnicas clássicas de otimização para transformar laços seqüenciais em laços paralelos. As técnicas principais foram:

a. Substituição do algoritmo seqüencial por outro paralelo;

b. Inversão de laços para permitir a vetorização na maior dimensão do array;

c. Quebra de grandes laços seqüenciais em seqüências de laços menores, muitos deles paralelos.

Outro fator relevante é quando a vetorização e paralelização OpenMP incidem no mesmo aninhamento. Nesse caso, utilizou-se a inversão de laços para gerar um aninhamento onde os laços internos gerassem o maior vetor possível, gerando paralelismo de primeiro nível (vetorização) nos laços internos e paralelismo de segundo nível (OpenMP) no laço externo.

Há casos de ganhos notáveis quando se aplica combinações das técnicas. Por exemplo, quando grandes aninhamentos de laços são quebrados em aninhamentos de laços menores que, por sua vez, sofrem inversão de laços. A figura 2 exemplifica esse procedimento.

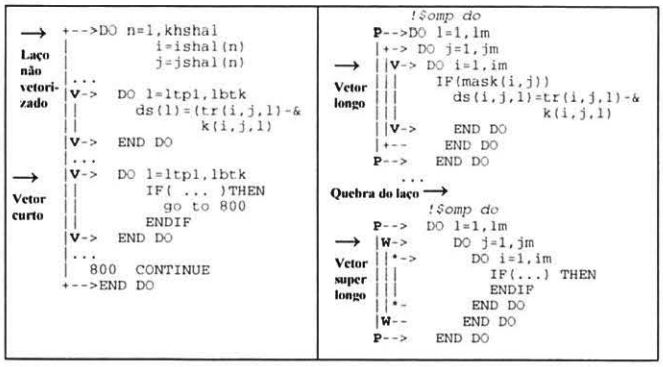


Figura 2. Aplicação das técnicas de otimização

4.3 Segundo nível de paralelização

A segunda etapa do trabalho foi a paralelização para memória compartilhada. Para tanto, utilizou-se uma metodologia de três passos:

a. Paralelização de granulação fina em laços que não invocam procedimentos e não requeiram reestruturação. Essa estratégia se aplica ao código todo, mas produz resultados modestos. Restringiu-se seu uso a laços com, no máximo, uma centena de comandos.

b. Paralelização de granulação grossa em laços que invocam procedimentos. Isto requer transformar os procedimentos em procedimentos puros [6]. Este passo, mais complexo que o anterior, exige grande esforço de implementação, visto que é necessário compreender o fluxo de dados do procedimento. Entretanto, o ganho computacional é significativo.

c. Reestruturação de laços para permitir a paralelização de granulação grossa. Consiste em utilizar a técnica clássica de fusão de múltiplos aninhamentos de laços em um único aninhamento. O corpo do laço mais externo do aninhamento único é transformado em um procedimento puro. O novo laço contém apenas a invocação do procedimento. Esse passo permite paralelizar o laço mais externo obtendo também um expressivo ganho computacional.

Esta metodologia ordena os passos em níveis crescentes de dificuldade e gera resultados ao fim de cada passo. A parte crítica do trabalho foi construir um novo nível de paralelismo (OpenMP) sobre um nível previamente existente (vetorização), sem que o novo nível elimine o anterior.

Nos raros casos em que e a vetorização atingia o laço intermediário de um aninhamento, que também era alvo de paralelismo, foi necessário alterar o *schedule* desse laço, dividindo-o em blocos que acessavam posições contíguas de memória, como mostra a Figura 3.

Vetorização em *i* e *j*	Vetorização em *i* e Paralelização em *j*	Vetorização em *i* e Paralelização em *j*
+->DO l=1,lm |v->DO j=3,jm-2 ||v->DO i=1,im ||| ... ||v->END DO |v->END DO +->END DO	!$omp parallel +->DO l=1,lm \| !$omp do |P-->DO j=3,jm-2 ||v-> DO i=1,im ||| ... ||v-> END DO |P-->END DO +->END DO !$omp end parallel	!$omp parallel +->DO l=1,lm \| !$omp do & | !$omp schedule(static,jm-4/np) \|P--->DO j=3,jm-2 \|\|v---> DO i=1,im \|\|\| ... \|\|v---> END DO \|P--->END DO +->END DO !$omp end parallel
Automática	**Distribuição cíclica (*ineficiente*)**	**Distribuição por blocos (*eficiente*)**

Figura 3. Paralelismo OpenMP sobre a vetorização

De forma geral, os ganhos em paralelismo OpenMP não foram obtidos a custa da redução de operações vetoriais.

5. Resultados obtidos

5.1 Desempenho

Um indicador de desempenho do primeiro nível de paralelismo (vetorização) é o aumento do tempo de execução vetorial (como fração do tempo total) que passou de 92% para 97,5%, o que é satisfatório. Entretanto, ainda há espaço para otimização. Já o tamanho médio dos vetores aumentou de 84 para 111 elementos,

que é insatisfatório face ao tamanho ideal dos vetores (pelo menos 256 elementos [5]).

O indicador de desempenho do segundo nível de paralelismo (OpenMP) é a aceleração (*speed-up*), definida como a razão entre o tempo de execução seqüencial e o tempo de execução paralelo do mesmo problema. Essa métrica depende do número de processadores utilizado – intuitivamente, deseja-se que um programa execute n vezes mais rapidamente se utilizarmos n processadores do que se utilizarmos um processador.

A Figura 4 apresenta a curva de *speed-up* de 1 a 8 processadores. O resultado mostra que o código executa 1,82 vezes mais rápido com 2 processadores, 3,04 vezes mais rápido com 4 processadores, 3,89 vezes mais rápido com 6 processadores e 4,55 vezes mais rápido com 8 processadores.

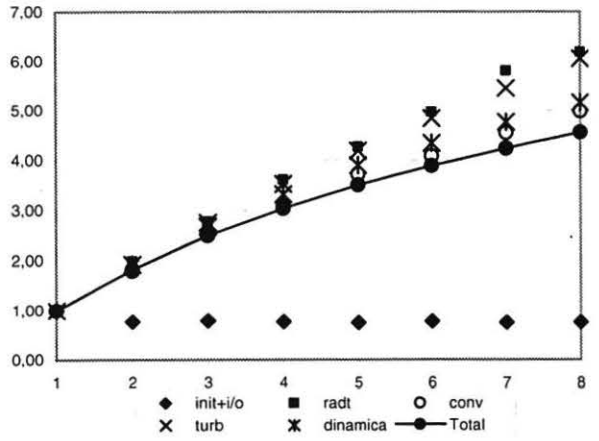


Figura 4. Desempenho do código paralelo OpenMP

Para avaliar se a aceleração é satisfatória, temos fatores críticos a considerar – em particular, quantificar a importância dos trechos do programa que ainda não foram paralelizados, o que determina a fração seqüencial do tempo de execução, segundo a Lei de Amdahl[7].

Por essa lei, o menor tempo de execução possível é o tempo da fração seqüencial, para qualquer número de processadores. Conseqüentemente, a aceleração está limitada à razão entre o tempo total e o tempo da fração seqüencial do programa, mesmo usando um número ilimitado de processadores. Como a fração seqüencial do código (inicialização e I/O) ocupa 4% do tempo total, a aceleração máxima teórica é 6,25 com 8 processadores (pois os 8 processadores podem acelerar 96% do tempo de execução do programa; dessa forma, o tempo de execução está limitado a 96%/8+4% do tempo seqüencial, ou seja, 16% do tempo seqüencial). Observa-se na figura 5, um crescimento da fração seqüencial do código (Init+I/O) com o aumento do número de processadores. Nessa ótica, a aceleração obtida com 8 processadores tornou-se razoável.

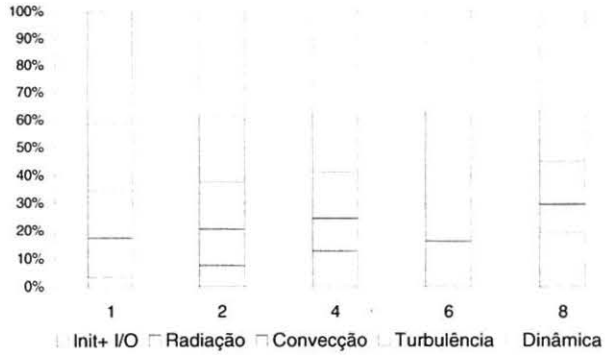


Figura 5. Medida de tempo por módulos

O ganho obtido pela combinação dos dois níveis de paralelismo é satisfatório. O tempo de processamento do código paralelo, rodando com 8 processadores, é 7,6 vezes mais rápido que a versão original em Fortran 77 (Tabela 1). Esse experimento foi realizado para previsão de 3 dias de integração, numa resolução de 40Km, com 38 níveis na vertical e sobre uma grade de 119 x 249 pontos, cujo domínio varia na longitude 35W a 85W e na latitude de 10N a 45S.

Tabela 1. Comparação entre as versões do código

Versão	Tempo (s)	Razão
Original	3967	7,60
Paralelo nível 1	2378	4,55
Paralelo nível 2 (8Proc)	522	1,00

5.2 Resultados Numéricos

Os resultados numéricos de programas vetorizáveis mudam com a opção de compilação empregada. Para um mesmo programa fonte, a opção de vetorização utiliza bibliotecas matemáticas distintas das utilizadas sem essa opção. Essas diferenças, em geral, são verificadas após a integração do modelo para um limiar de tempo. Obtive-se diferenças de arredondamento inferiores a 0,6% entre os resultados de execuções escalares e vetoriais para previsões de 24hs. Os resultados obtidos foram validados através da análise detalhada do comportamento dos campos meteorológicos.

O código resultante possui compatibilidade binária, ou seja, para cada conjunto de opções de compilação o resultado numérico (binário) não muda com o número de processadores, o que é notável.

5.3 Memória

No paralelismo OpenMP, há áreas de trabalho locais a cada processador. Logo, o aumento do número de processadores acarreta aumento da memória total

utilizada. Admitindo que a memória dependa linearmente do número de processadores, aproxima-se essa função por mínimos quadráticos sobre os dados experimentais, obtendo M = 19,45*P + 335,75, onde M é a quantidade de memória em MBytes e P o número de processadores utilizados.

Resumindo, o aumento do número de processadores acarreta aumento do uso de memória, à razão de 5,79% por processador, na resolução utilizada. Este custo foi considerado aceitável face à redução do tempo de execução (Tabela 2).

Tabela 2 . Uso de memória

Processadores	Memória (MB)
Original	757
1	354
2	376
4	414
6	452
8	491

A versão paralela requer de 46,4% a 59,7% da memória utilizada pela versão original. Essa redução deve-se à alocação dinâmica de memória, nativa no paralelismo OpenMP e inexistente na versão original. Ou seja, a versão original aloca toda a memória no início da execução, mantendo-a alocada durante toda a execução, enquanto a versão paralela acrescenta e retorna memória conforme a necessidade do procedimento que está sendo executado no momento. Em conseqüência, a cada instante da execução original, da ordem de 50% da memória alocada não está sendo utilizada.

6. Conclusão

Este trabalho buscou disponibilizar um código mais moderno e computacionalmente eficiente, que explore ao máximo as características da máquina.

Os objetivos foram alcançados de forma satisfatória. A otimização seqüencial exigiu mudanças razoáveis no algoritmo para permitir a vetorização e o melhor aproveitamento dos *pipelines*. A mudança na estrutura de dados para permitir a paralelização em dois níveis exigiu um grande empenho do grupo de desenvolvimento do projeto.

A paralelização OpenMP também foi satisfatória em virtude dos resultados apresentados. Durante o desenvolvimento do projeto houve a preocupação de garantir a portabilidade do código para máquinas de memória central.

O esforço de minimizar regiões paralelas abertas, gerou um paralelismo eficiente, mas ainda permite melhorias no desempenho computacional. Um fator importante quando aplicamos o paralelismo de dados no código, é o razoável balanceamento de carga obtido. Outro resultado importante foi que a variação do número de processadores não tem influência sobre o resultado dos campos meteorológicos.

Finalmente, é forçoso reconhecer a importância da primeira fase dos trabalhos – a modernização da codificação do programa simplificou, ao extremo, a aplicação das técnicas de paralelismo. Seria muito difícil, por exemplo, determinar quais variáveis são passíveis de replicação em um aninhamento de laços a paralelizar (ou seja, tornando-as *private*) sem saber rapidamente se a variável é local ou global.

7. Referências

[1] F. Mensinger, Z. I. Janic, S. Nickovic, D. Gavrilov, E D. G. Deaven, 1988: The step-mountain coordinate: Model description and performance for cases of Alpine lee cyclogenesis and for a case of Appalachian redevelopment. Mon. Wea. Rev., 116, 1493-1518.

[2] T. L. Black, 1994: The new NMC mesoscale Eta model: Description and forecast examples. Wea. and Forecasting, 9, 265-278.

[3] OpenMP Architecture Review Board. OpenMP Fortran application program interface: version 1.1. 1999. [S.l.:s.n] 76p.

[4] S. Nikovic, et al. Scientific documentation of the Eta model. Belgrade: [s.n]. 1998. 177p.

[5] NEC. Sx series applications training. 1998. [S.l.:s.n]

[6] Ansi Accredited Technical Subcommittee X3J3. International. standard progr. Fortran (Fortran 95). 1996.

[7] K. Doud, C. R. Severance, High performance computing. 2nd. ed. Sebastopol: O'Reilly, 1998.446p.