Estudo de Desempenho da Paralelização Automática com PLUTO e C em Plataformas Multicores e Gpu

Daniel Alves Paraiso[1], Augusto Brandão D'Oliveira[2] e Solon Venâncio de Carvalho[3]

[1] Programa de Mestrado ou Doutorado em Computção Aplicada – CAP
Instituto Nacional de Pesquisas Espaciais – INPE

[2] Centro de Previsão de Tempo e Estudos Climáticos – CPTEC
Instituto Nacional de Pesquisas Espaciais – INPE

[3] Laboratório Associado de Computação e Matemática Aplicada – LAC
Instituto Nacional de Pesquisas Espaciais – INPE

daniel.paraiso@lit.inpe.br, augusto.oliveira@cptec.inpe.br,
solon@lac.inpe.br

Abstract. *We present results of the automatic parallelization of a free software that calculates the Haralick texture features for each window that runs through an image. The automatic parallelization was done using the PLUTO code. Results of measuring the execution time are presented for multicore architectures and CUDA.*

Resumo. *Apresenta-se resultados da paralelização automática de um código livre desenvolvido em linguagem C para calcular as características de textura de Haralick para cada janela que percorre uma imagem. A paralelização automática foi feita usando o código PLUTO. Resultados da medida do tempo de execução são apresentados para arquiteturas multicore e CUDA.*

Palavras-chave: *PLUTO, Haralick, paralelização automática,CUDA, multicores.*

1. Introdução

O processo de escrever códigos para aproveitar o paralelismo disponível nas modernas CPU multicores, por exemplo, programando em OpenMP, bem como nas centenas de cores existentes nas GPU é árduo, e nem sempre acessível ao engenheiro ou pesquisador que deseja acelerar a execução de suas aplicações.

O uso de ferramentas automáticas para paralelização, apresenta uma possibilidade de solução sem necessariamente ter que aprofundar-se nas complexidades da programação paralela usando OpenMP, ou da tecnologia CUDA, um desses programas é o PLUTO.

O software PLUTO é uma ferramenta de paralelização automática, com base no modelo poliédrico. O modelo poliédrico para otimização do compilador fornece uma abstração para realizar transformações de alto nível, como otimização e paralelismo em laços aninhados e afins. PLUTO transforma programas fonte em linguagem C para programas fonte com paralelismo de granularidade grossa. A referência para algoritmo de escalonamento usado por PLUTO foi publicada por Bondhugula et al em 2008[1].

Códigos paralelos usando OpenMP para multicores podem ser gerados automaticamente

a partir de programas sequenciais em C. Paralelização externa, interna, ou pipeline é alcançada (puramente com programas OpenMP).

Uma avaliação experimental e comparação com as técnicas anteriores podem ser encontrados em Bondhugula [2]. Embora a ferramenta seja totalmente automática (C para C com OpenMP), um número de opções são fornecidas (ambos da linha de comando e através de arquivos de meta) para sintonizar aspectos como tamanhos de azulejos, fatores de unroll, e a estrutura de fusão exterior loop. Cloog-ISL é usado para geração de código. Uma versão com suporte para geração de código CUDA também foi utilizada.

O uso de paralelização automática usando C e PLUTO para plataformas multicores e GPU tem inúmeras aplicações no processamento de imagens médicas. Os descritores de Haralick podem ser usados para determinação automática de pedras de rins, tumores em mamografias, etc. Neste trabalho propõe-se um método de paralelização híbrido que combina PLUTO com trechos paralelizados de maneira convencional.

2. Método

Fez-se um estudo do desempenho de dois programas em C, um que implementa o algoritmo de Gauss-Seidel, e outro um código desenvolvido nesta pesquisa que possibilita calcular descritores de Haralick a partir da matriz de co-ocorrência, para fins de detecção automática de regiões com características pré determinadas.

Dado o código em C, basta colocar as diretivas “#pragma scop” e “#pragma endscop”, respectivamente, antes e depois da seção que vai ser paralelizada por PLUTO. A figura 1 apresenta a parte do código referente ao algoritmo de Gauss-Seidel, paralelizada com PLUTO.

```
#pragma scop
   for (t=0; t<=T-1; t++)  {
      for (i=1; i<=N-2; i++)  {
         for (j=1; j<=N-2; j++)  {
            a[i][j] = (a[i-1][j-1] + a[i-1][j] + a[i-1][j+1]
                  + a[i][j-1] + a[i][j] + a[i][j+1]
                  + a[i+1][j-1] + a[i+1][j] + a[i+1][j+1])/9.0;
         }
      }
   }
#pragma endscop
```

Figura 1 – Paralelização com PLUTO da parte do código referente ao algoritmo de Gauss-Seidel.

O primeiro programa é chamado “seidel”, e é uma adaptação do código exemplo que vem com o software PLUTO. Neste programa acrescentou-se um segundo laço aninhado de paralelização utilizando apenas OpenMP para os testes de códigos híbridos (OpenMP e PLUTO), mostrada na Figura 2. A linha: “sum += a[i][j] * a[i][j];” calcula o somatório dos quadrados de uma matriz, que poderia ser uma matriz de co-ocorrência(matriz utilizada para todos os cálculos das características de haralick). Esta linha, somatório da figura 2, foi alterada repetidas vezes para fazer as experiências mostradas nas Tabelas 5, 6 e 7, para investigar a capacidade de OpenMP lidar com somatórios duplos, bem como a capacidade de

PLUTO lidar estes somatórios na presença de funções logaritmos e valores absolutos que aparecem em muitas aplicações, em especial no cálculo das características de textura propostas por Haralick, que foi uma das motivações para este estudo. Constatou que PLUTO na versão 0.8 não aceita paralelizar loops com logaritmos, e funções transcendentais.

```
#pragma omp parallel private(j)
{
 #pragma omp for schedule(dynamic, CHUNKSIZE)
      for (i = 0; i < N; ++i)
         for (j = 0; j < N; ++j)
            sum += a[i][j] * a[i][j];
}
```

Figura 2 – Paralelização com OpenMP da parte do código referente ao duplo somatório.

O segundo programa é chamado de “haralick”, e tem basicamente 3 blocos, um para calcular a matriz de co-corrência, outro para calcular as cinco características de Haralick: ASM, contraste, homogeneidade, soma da média e soma da variância, e o terceiro bloco contendo as demais características.

Dado o código em C, foi colocado as diretivas “#pragma scop” e “#pragma endscop”, respectivamente, no início e no final da seção que vai ser paralelizada por PLUTO. Fizemos um estudo de várias possibilidades de paralelização híbrida, paralelizando um bloco do programa usando PLUTO e outro bloco usando OpenMP convencional, contendo múltiplos loops.

Para os testes foi utilizado duas máquinas: a primeira máquina com CPU AMD Phenom X6, (hexacore), e uma placa de vídeo de arquitetura FERMI, a GTX550 da NVIDIA, com CUDA Toolkit 4.0 e segundo computador, com processador I3 M370 quadcore. A segunda máquina foi utilizada apenas para testes com OpenMP.

3. Resultados

A Tabela 1 mostra os resultados obtidos para programa seidel, e a Tabela 2 os resultados para o programa haralick utilizando OpenMP e PLUTO. No caso do programa seidel, a compilação com PLUTO permitiu uma paralelização automática bastante eficiente, e inclusive usando CUDA. Já para haralick, os resultados de PLUTO revelaram um ganho de apenas cerca de 10% utilizando OpenMP. Embora simples de se obter, os resultados mostram um ganho bastante pequeno.

Portanto decidiu-se não usar paralelização automática para o cálculo das características de Haralick de textura utilizando PLUTO e pela técnica híbrida proposta neste trabalho. Como podemos ver claramente nos resultados da Tabela 3, no caso de Haralick, este estudo indicou a necessidade de usar-se programação C e OpenMP convencional, e esta será a continuação lógica desta pesquisa.

Tabela 1: Tempos medidos para o programa seidel. Usando-se o Phenom II

Seidel	Phenom II 6	GT550TL CPU + GPU
C serial	9,5 s	-
C OpenMP	2,2 s	-
C PLUTO OpenMP	2,5 s	-
C PLUTO CUDA	-	1,30 + 0,62 s

Tabela 2: Tempos medidos para o programa haralick, para o processamento de uma só imagem, no X6.

Haralick	Phenom II 6
C serial	0,95 s
C OpenMP PLUTO	0,84 s

Tabela 3: Tempos medidos para o programa haralick, para o processamento de dez imagens no I3.

Haralick	I3 370 M
C - serial	6,96 s
C – OpenMP(PLUTO) N=1	5,94 s
C – OpenMP(PLUTO) N=2	5,82 s
C – OpenMP(PLUTO) N=3	5,83 s
C – OpenMP(PLUTO) N=4	5,97 s

As Tabelas 4, 5 e 6 mostram resultados de medidas de tempo para várias funções e hibridizações. Não houve resultados significativos da implementação híbrida somatórios em funções logaritmos e valores absolutos que aparecem em muitas aplicações (tabela 5). Observe-se o speed-up e eficiência da tabela 6 em relação ao número de processadores, na figura 3 e 4, respectivamente. Vê-se claramente que o ganho de tempo obtido com o uso da paralelização híbrida em somatórios (tabela 6) é mais substancial em relação ao tempo de processamento serial e OpenMP com PLUTO, apesar da sua simplicidade de implementação.

Tabela 4: Tempos medidos para o processamento no I3 do somatório duplo, com i e j variando de 1 ate 1000 para as funções indicadas. Foram usados um programa serial em C, e sua versão paralelizada usando-se OpenMP convencional

	$\sum_i \sum_j [(i+j)(i+j)]$	$\sum_i \sum_j [\exp(i+j)]$	$\sum_i \sum_j [\log(i+j)]$
C serial	0,38 s	2,76 s	4,49 s
C – OpenMP N=4	0,33 s	1,53 s	2,18 s
C – OpenMP N=3	0,37 s	1,68 s	2,53 s
C – OpenMP N=2	0,41 s	1,92 s	3,14 s
C – OpenMP N=1	0,47 s	2,83 s	4,62 s

Tabela 5: Tempos medidos para o processamento no I3 .Foram usados um programa serial em C, e sua versão paralelizada usando-se PLUTO e uma versão de paralelização hibrida, metade PLUTO e metade feita com OpenMP convencional

	PLUTO em todo programa			Hibrido: OpenMP e PLUTO		
	ΣΣabs(i-j)	ΣΣexp(i+j)	ΣΣlog(i+j)	ΣΣabs(i-j)	ΣΣexp(i+j)	ΣΣlog(i+j)
C serial	8,42 s	8,43 s	8,44 s	8,42 s	8,43 s	8,44 s
N=4	3,01 s	3,07 s	3,09 s	3,01 s	3,02 s	3,05 s
N=3	3,76 s	3,82 s	3,83 s	3,76 s	3,77 s	3,81 s
N=2	4,62 s	4,98 s	4,66 s	4,62 s	4,63 s	4,64 s
N=1	8,98 s	8,98 s	9,01 s	8,93 s	9,01 s	8,97 s

Tabela 6: Tempos medidos para o processamento no I3 .Foram usados um programa serial em C, e sua versão paralelizada usando-se PLUTO e uma versão de paralelização hibrida, metade PLUTO e metade feita com OpenMP convencional

	PLUTO	Híbrido	PLUTO	Híbrido
Número de processadores	$\sum_k \sum_i \sum_j [a(i-1,j)/k + a(i+1,j)/k]$		$\sum_i \sum_j [P(i,j)\ P(i,j)]$	
N=4	9,71 s	7,53 s	3,25 s	3,02 s
N=3	10,74 s	8,29 s	4,01 s	3,75 s
N=2	12,62 s	9,15 s	4,83 s	4,61 s
N=1	17,09 s	18,01 s	9,22 s	8,94 s
Serial	17,47 s		8,41 s	

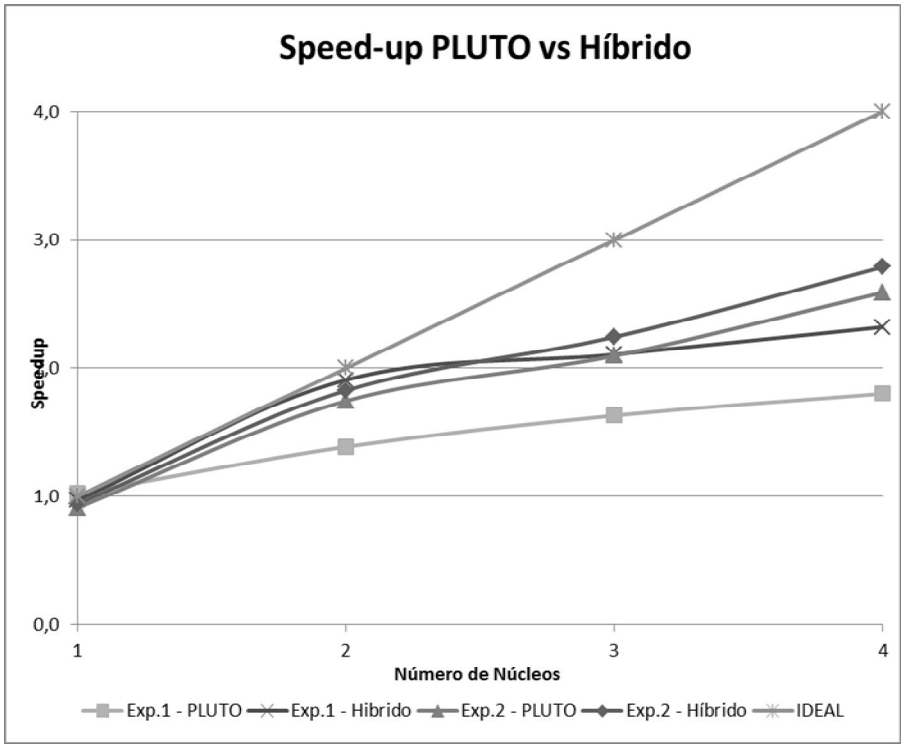

Figura 3 – Gráfico de Speed up da paralelização com PLUTO e da versão híbrida de PLUTO e OpenMP.

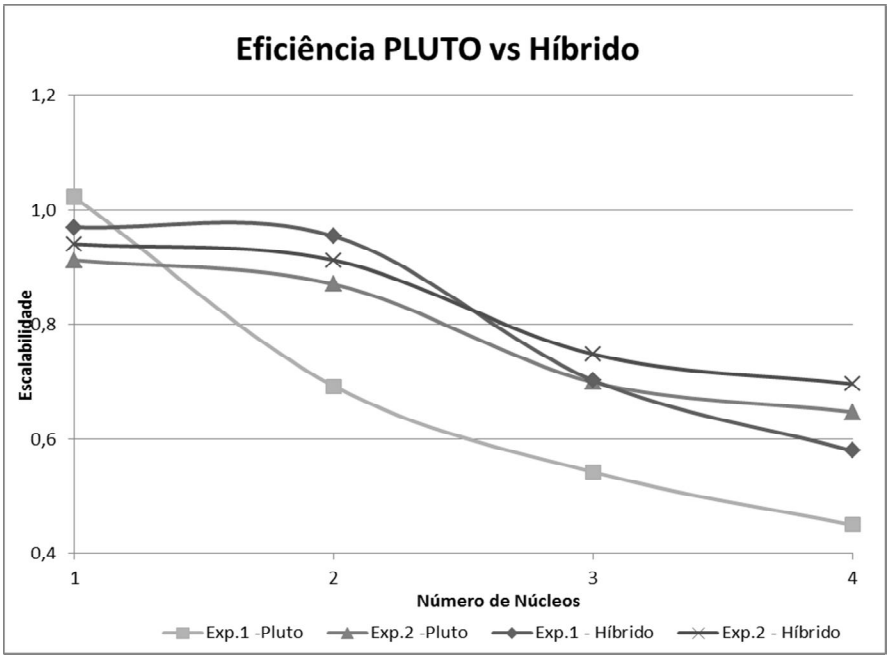

Figura 4 – Gráfico de eficiência da paralelização com PLUTO e da versão híbrida de PLUTO e OpenMP.

4. Conclusões

Os resultados mostrados levaram à conclusão de que para o programa haralick não compensa usar paralelização automática e estão em andamento esforços para a produção de uma versão escrita em OpenMP C e outra em CUDA C, para obter resultados melhores.

Os resultados obtidos com seidel mostram, entretanto que para algoritmos simples que não usam funções matemáticas como valor absoluto e logaritmos e com duplos somatórios, por exemplo, os resultados são bastante bons, e podem ser usados com ganho devido a sua facilidade de implementação.

Os resultados também mostraram que em alguns casos como os mostrados acima é vantajoso usar-se uma paralelização híbrida, onde parte do programa é paralelizada por PLUTO devido à sua simplicidade de uso, e a parte que PLUTO não aceita paralelizar pode então ser tratada no todo ou em parte, com os métodos convencionais de programação OpenMP. Principalmente ao lidar com somatórios duplos esta técnica híbrida é vantajosa.

Esses resultados mostram que em alguns casos podemos usar PLUTO para paralelizar códigos, de maneira simplificada, acessível a não especialistas e assim aproveitar melhor a potência computacional que está disponível em máquinas multicore, e também em computadores equipados com GPU, usando tecnologia CUDA. Às aplicações dessas técnicas a computação em diversas áreas.

5. Agradecimentos

Agradecemos a CAPES e ao CNPq, pelo apoio financeiro.

6. Referências

[1] BondhugulaU, Baskaran M, Krishnamoorthy S, Ramanujam J,Rountev A, Sadayappan P. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. International Conference on Compiler Construction (ETAPS CC), Apr 2008, Budapest, Hungary.

[2] Bondhugula U, Hartono A, Ramanujan J, Sadayappan P. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer. ACM SIGPLAN Programming Languages Design and Implementation (PLDI), Jun 2008, Tucson, Arizona.