Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática
Programa de Pós-Graduação em Computação

Segmentação de imagens em paralelo para uso em um sistema com múltiplos elementos de processamento

por

ANDRÉ BORIN SOARES

RELATÓRIO TÉCNICO

Altamiro Amadeu Susin
Orientador

Porto Alegre, 16 de novembro de 2004

LISTA DE ABREVIATURAS

P.C.	Processador central
NoC	Rede no Chip (*Network on Chip*)
P.E.	Elemento de Processamento (*Processing element*)

LISTA DE SÍMBOLOS

M_i	número de divisões da imagem na direção horizontal
N	número de divisões da imagem na direção vertical
P	percentual de tempo necessário para a realização da computação pelo sistema

LISTA DE FIGURAS

Figura 1 Grafo de comunicação genérico para o problema de segmentação de imagens, representando um número arbitrário de processadores.

1 INTRODUÇÃO

O uso de métodos de inspeção visual e processamento de imagens em aplicações diversas tem crescido bastante nos últimos anos em decorrencia do significativo aumento da capacidade computacional aliada a popularização de câmeras digitais. Contudo, para muitas destas aplicações o desempenho de um único processador de prateleira ainda não é o suficiente. Em outros casos, o desempenho é aceitável, mas por apresentar um consumo de potência elevado, não torna-se prático quando este sistema utiliza baterias (sistema embarcado). Nestes casos, existe a necessidade de realizar uma exploração do espaço de projeto para buscar a alternativa que satisfaz os requisitos de desempenho e potência.

Este trabalho descreve uma aplicação que será utilizada como estudo de caso, e um algoritmo parametrizável criado para ser utilizado em uma exploração de espaço de projeto, permitindo a determinação do número de elementos de processamento.

1.1 PROBLEMA

Dada uma imagem binária, com pixels que representam o fundo com valor 0 e pixels que representam os objetos com valor 1, gerar uma imagem com os pixels de cada objeto identificados com um valor. Este valor será um número associado ao objeto.

Deseja-se que a solução proposta permita resolver o problema de forma paralela e com a exploração do paralelismo parametrizável.

2 SOLUÇÕES PROPOSTAS

2.1 VERSÃO SERIAL

Propõe-se inicialmente realizar uma varredura na imagem, marcando em cada pixel se este já foi visitado.

Quando se tratar de um pixel pertencente ao fundo, nenhuma informação é adicionada ao pixel. No caso de se encontrar um pixel de um objeto, este é marcado com o número de identificação do objeto, que é iniciado em 1 e incrementado a cada objeto.

Um objeto é completamente marcado quando o primeiro pixel é encontrado. Durante a marcação, após um pixel do objeto ser verificado, seus 4 vizinhos são também verificados, e assim sucessivamente, de forma recursiva. Se o pixel sendo verificado pertencer ao objeto, mas já tiver sido visitado, ou se for um pixel representando o fundo, nenhuma informação é adicionada e seus vizinhos não são visitados. Se o pixel não tiver sido visitado, é adicionada a informação do número do objeto e os vizinhos são verificados. Este processo ocorre de forma recursiva, até que todos os pixels do objeto tenham sido verificados. Neste ponto, a varredura prossegue até o fim da imagem, verificando se há outros objetos.

2.2 VERSÃO PARALELA

Com a finalidade de verificar se é possível explorar de forma eficiente o paralelismo presente na solução proposta foi desenvolvida uma segunda solução paralela. Nesta versão, a imagem é dividida em M x N sub-imagens. Em cada uma das sub-imagens é executado o algoritmo da versão serial. Contudo, após esta etapa, cada sub-imagem terá uma numeração própria, pois supõe-se que não há como saber a priori quantos objetos existirão em cada sub-imagem. Poderão existir também objetos nas fronteiras entre duas ou mais sub-imagens, os quais devem ser identificados corretamente como apenas um objeto.

A determinação das intersecções é realizada após o processamento de cada sub-imagem utilizando a versão serial. A coluna da direita de uma sub-imagem é comparada com a coluna da esquerda da sub-imagem da sua direita(quando existir). A linha inferior de uma sub-imagem também é comparada com a linha superior da sub-imagem de baixo (quando existir). Se houverem dois pixels na mesma coluna diferentes do fundo (quando estiver verificando as linhas) ou dois pixels na mesma linha (quando estiver verificando as colunas), se considera que uma intersecção foi encontrada.

Optou-se por gerar um algoritmo que, dado uma lista de pares indicando as correspondencias de dois objetos em duas sub-imagens vizinhas com um único objeto e o número de regiões de cada sub-imagem, gera uma numeração única para todas as sub-imagens. Esta numeração única é então substituida em cada região para produzir uma imagem final com numeração única dos objetos.

Este algoritmo funciona da seguinte maneira:

1-É criada uma lista (lista A) de todos os objetos de todas as sub-imagens. Esta lista retém a informação de qual sub-imagem é a detentora do objeto.

2-É criada uma lista (lista B) de todas as intersecções de objetos entre as sub-imagens vizinhas. Esta lista retém a informação de quais são as duas sub-imagens as quais se refere cada intersecção

3-Substitui-se na lista A o identificador de cada objeto por um identificador global único.

4-A cada substituição na lista A, verifica-se na lista B quais os elementos da lista que se referem a mesma sub-imagem e possuem a mesma numeração e substitui-se estes elementos pela numeração global.

5-De posse das duas listas já com uma numeração global, inicia-se o processo de eliminação de intersecções. Para cada elemento da lista B escolhe-se arbitrariamente um dos elementos do par e substitui-se nas duas listas este elemento pelo outro elemento do par. Este processo equivale a unir os objetos partidos entre duas ou mais sub-imagens. A substituição na lista B garante que não haverão substituições incorretas (ou seja, elimina-se uma referência adicional a um mesmo objeto e impede-se que esta referência seja utilizada em futuras substituições).

6-Ao final do processo, tem-se na lista A uma lista com todos os objetos de todas as imagens com a numeração global. Se diferentes objetos dentro da região tiverem um mesmo número global significa que são partes do mesmo objeto, unidas em diferentes sub-imagens.

7-O último passo consiste em percorrer a lista A substituindo a numeração resultante, que pode não ser uniforme por uma numeração uniforme. Isto é feito montando-se uma tabela de substituição, adicionando-se um novo elemento a tabela cada vez que um objeto diferente for encontrado.

8-A numeração final é substituida em cada sub-imagem.

3 IMPLEMENTAÇÃO FINAL

A versão final da aplicação SegImage foi descrita em C++. Cada tipo de componente do sistema, é representado por uma classe. A comunicação entre tarefas é realizada através de funções especializadas, que representarão trocas de mensagens. Utilizou-se funções de recebimento de dados, que são chamadas pelo componente que deseja enviar a mensagem. Assim, quando a aplicação inicia a sua execução, é necessário passar um ponteiro para referenciar o objeto de destino, de forma que o componente que deseja enviar a mensagem possa chamar a função no objeto de destino. Desta forma o envio de uma mensagem fica explícito e pode-se simplesmente substituir esta chamada de função no objeto de destino por uma função do tipo *envia*. Isto será útil se a arquitetura for executada em um hardware com suporte a função send, e não mais uma simulação.

Foi criada uma classe para o elemento de processamento, responsável pelo processamento de uma sub-imagem. Os dados mais importantes a serem armazenados nesta classe são a matriz que armazena a imagem de entrada, a que armazena a imagem de saída e a pilha que armazena as coordenadas durante o preenchimento de um objeto. As principais funções são visitar um pixel, que trata da marcação com um número de objeto e a visita a pixels vizinhos, a função de processamento de fronteiras e as funções de recebimento de mensagens.

Foi criada também uma classe para o processador central, responsável pela geração da numeração global única. As principais funções são as responsáveis pela montagem da lista de intersecções e função de montagem da tabela de saída.

A divisão da imagem em sub-imagens é feita através da divisão da imagem na direção horizontal (M colunas) e na direção vertical (N linhas). Valores potências de 2 produzem um funcionamento correto. Pode-se utilizar no caso extremo apenas um elemento de processamento, e neste caso não é necessário o uso do processador central, uma vez que não existem intersecções de objetos, pois não existem sub-imagens.

3.1 VANTAGENS PARA A IMPLEMENTAÇÃO EM HARDWARE

Da forma como foi proposta a solução do problema, foram atingidos os objetivos de exploração do paralelismo através da possibilidade de execução da etapa de rotulação em paralelo nas M x N sub-imagens.

Pode-se supor que a imagem inicial encontra-se armazenada em uma memória principal e que cada sub-imagem será processada em um núcleo com uma memória local. Assim, tem-se para uma imagem dividida em M x N sub-imagens, M x N núcleos ou elementos de processamento. Uma boa forma de interligar estes núcleos é através da utilização de uma Rede no Chip ou NoC (Network on Chip). A utilização de tal rede permite também a interligação da matriz de M x N elementos de processamento ou P.E.s (processing elements) com elementos adicionais no sistema, como a memória principal e um processador central onde será determinada a numeração única. Esta mesma rede permite que os núcleos troquem informações sobre linhas e colunas para determinação das intersecções.

3.2 TAXAS DE COMUNICAÇÃO ENTRE OS NÚCLEOS DE PROCESSAMENTO

Com a finalidade de modelar a aplicação para realizar um posicionamento automático em uma NoC, foi calculada a taxa de comunicação entre os diferentes elementos do sistema. Na Tabela I encontra-se um cálculo do volume de dados, em bytes, transmitido em cada quadro de uma

imagem de dimensões 640x 480 pixels. Este valor deve ser dividido pelo produto do número de quadros por segundo pela fração do tempo que ocupa cada transferência. A soma do percentual de todas as transferências que ocorrem em série para cada quadro não pode ultrapassar (100-P)%, onde P é o percentual de tempo necessário para a realização da computação.

Tabela I: Volume de dados a ser transferido pela aplicação de segmentação de imagens, considerando uma imagem de 640x480 bytes e apenas um quadro.

PA ↔ ME		Número de bytes total da imagem (n_{BI}) = 640x480 = 307200 bytes
		Número de bytes de cada segmento (n_{BS}) = n_{BI} / n_{PE} = 307200 / 4 = 76800 bytes (considerar 2x – ida e volta)
PA ↔ PA	Em X	Número de bytes da imagem em Y (n_{BY}) = 480 bytes
		Número PAs em Y (n_{PAY}) = 2
		Número de bytes na fronteira de cada PA para Y (n_{BPY}) = n_{BY} / n_{PAY} = 480 / 2 = 240 bytes
	Em Y	Número de bytes da imagem em X (n_{BX}) = 640 bytes
		Número PAs em X (n_{PAX}) = 2
		Número de bytes na fronteira de cada PA para X (n_{BPX}) = n_{BX} / n_{PAX} = 640 / 2 = 320 bytes
PA ↔ PC		Número de bytes de controle de vizinhança (n_{CY}) = 128 bytes (estimado com base no tamanho dos objetos em imagens típicas, equivale a 64 pares)

3.3 GRAFO DE COMUNICAÇÃO DA APLICAÇÃO

Pode-se extrair um grafo da aplicação descrita considerando-se cada elemento do sistema como sendo um nodo e usando uma aresta para ligar elementos que sem comunicam. Adicionalmente, pode-se adicionar um valor à cada aresta do grafo, sendo que este valor corresponde ao volume de dados transferido entre os dois nodos.

Tal grafo, considerando-se um número arbitrário de processadores, pode ser observado na Fig. 1. Não foi incluido nenhum valor de volume de dados transferido porque estes valores são particulares para uma dada configuração.

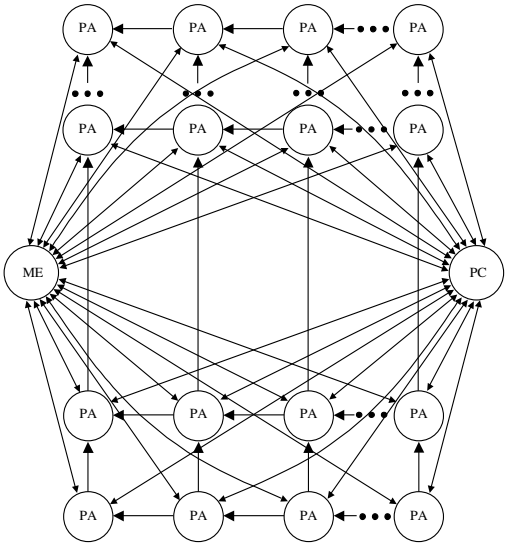


Figura 1: Grafo de comunicação genérico para o problema de segmentação de imagens, representando um número arbitrário de processadores.

Tarefas que são executadas em cada um dos elementos de processamento

Na Tabela II estão listados os passos que são executadas nos elementos do sistema.

Tabela II – Passos executados nes elementos do sistema

Processamento da sub-imagem	Eliminação de redundâncias
P_1: Recebe imagem da ME P_2: Processa imagem gerando numeração local P_3: Se (tem P.E. vizinho esquerdo) Envia fronteira p/ P.E. esquerdo P_4: Se (tem P.E. vizinho acima) Envia fronteira para PE acima P_5:Se tem vizinho direito, recebe coluna P_6:Se tem vizinho abaixo, recebe linha P_7:processa fronteiras P_8: Envia cv para PC P_9: Recebe cv do PC P_{10}: Processa imagem P_{11}: Envia imagem para ME	P_1: Enquanto (não terminar todos PAs) Recebe cv de cada P.E. P_2: Elimina redundância dos objetos e mapeia numeração local para numeração global P_3:Envia nova numeração para os objetos
(a) Legenda: cv – controle de vizinhança	(b)

4 EXEMPLO

A aplicação SegImage está exemplificada aqui com uma imagem de 640x480, uma taxa de processamento de 15 quadros/seg, e 4 P.E.s. Assim, o número de pixels em X é 640 e o número de pixels em Y é 480. Como existem 2 P.E.s para cada dimensão, o número de bytes que cada P.E. deve processar na fronteira em X é 320 e o número de bytes que cada PA deve processar na fronteira em Y é 240 (sub-imagem de 320 x 240). As características da imagem e cada segmento são apresentadas na Figura 2.

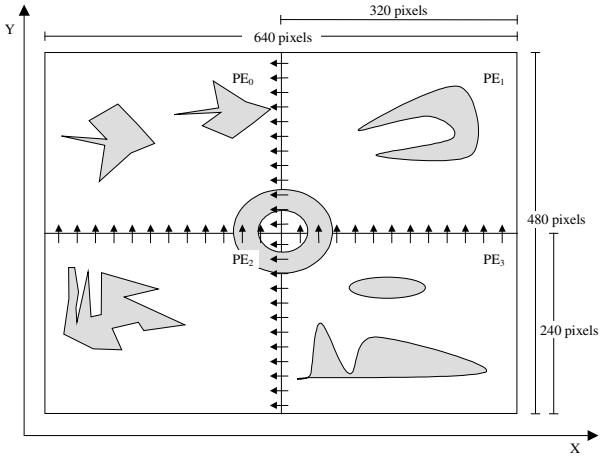
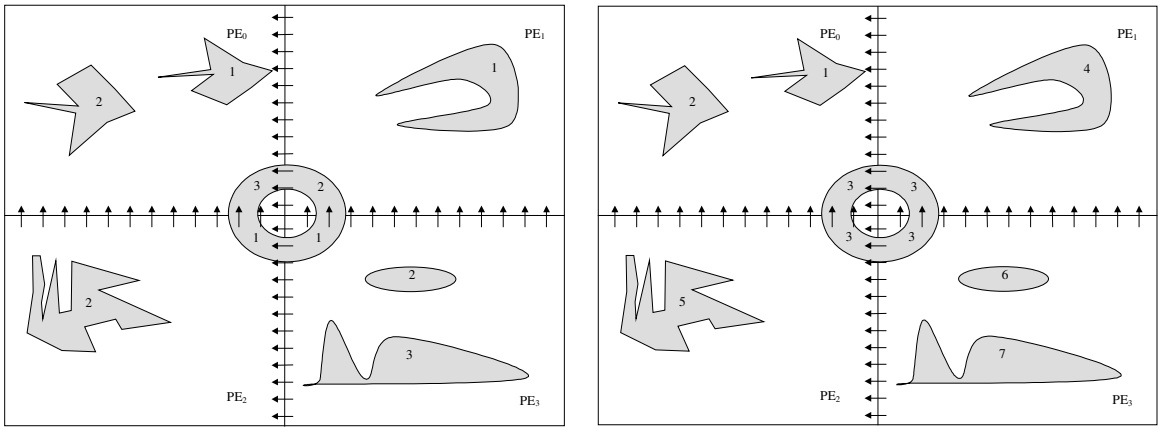


Figura 2: Segmentação da imagem avaliada com os correspondentes PAs.

Na Fig. 3 estão apresentadas duas imagens e os objetos com a respectiva numeração. Pode-se observar na Fig. 3(a) que em cada sub-imagem é produzida uma numeração independente, que é incrementada segundo a ordem dos topos dos objetos e da borda esquerda, como conseqüência do processo de varredura. A eliminação da redundância na numeração dos objetos é eliminada com auxílio do algoritmo descrito, resultando na numeração presente na Fig. 3(b).



(a) (b)

Figura 3 – (a) Numeração inicial com redundância entre sub-imagens; (b) Numeração final após eliminação de redundâncias

Pode-se observar na Tab. III os diversos passos executados no algoritmo de eliminação de redundâncias. Na Tab. III(a) encontra-se a lista A e a lista B, montadas de acordo com o método já descrito anteriormente. A lista A contém os objetos de cada sub-imagem, ordenados por objeto e por sub-imagem. Na Tab. III-lista B, o símbolo P é utilizado para indicar o objeto principal e S é utilizado para indicar o objeto secundário, ou seja, o objeto principal que foi detectado como outro objeto na sub-imagem vizinha. Cada número em S está acompanhado do símbolo h para indicar que o vizinho considerado é o da direção horizontal e v para indicar que é o vizinho da direção vertical. No passo 1(Tab. III(b)), a numeração local é substituida por uma numeração global, mantendo-se a redundância dos objetos mas corrigindo-se a coluna S da lista B. Nos passos 2-5, é realizada a substituição da numeração dos objetos, seguindo a ordem das linhas da lista B e substituindo-se em toda a lista A e no restante das linhas da lista B. Na Tab. III(g) encontra-se o resultado final, obtido a partir da substituição da numeração da lista A da Tab. III(f) por uma numeração uniforme.

Tabela III – Passos para a eliminação de redundância dos objetos do exemplo da Fig.3. Na lista B, P é o objeto principal e S é o objeto secundário (mesmo objeto considerado como outro objeto). A Tabela II(a) refere-se ao estado inicial das 2 listas; (b)-(f) são os passos intermediários e (g) é o estado final das duas listas.

Início			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	3	2h
2	0	3	1v
3	0		
1	1	2	1v
2	1		
1	2	1	1h
2	2		
1	3		
2	3		
3	3		

(a)

Passo 1			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	3	5
2	0	3	6
3	0		
4	1	5	8
5	1		
6	2	6	8
7	2		
8	3		
9	3		
10	3		

(b)

Passo 2			
Lista A		ListaB	
Obj.	P.E	P	S
1	0	-	-
2	0	3	6
3	0		
4	1	3	8
3	1		
6	2	6	8
7	2		
8	3		
9	3		

(c)

Passo 3			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	3	8
3	1		
3	2	3	8
7	2		
8	3		
9	3		
10	3		

(d)

Passo 4			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-

Passo 5			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-

Fim			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-

3	1		
3	2	3	3
7	2		
3	3		
9	3		
10	3		

(e)

3	1		
3	2	-	-
7	2		
3	3		
9	3		
10	3		

(f)

3	1		
3	2	-	-
5	2		
3	3		
6	3		
7	3		

(g)

Na Fig. 4, pode-se observar o grafo de comunicação para o exemplo apresentado, para uma implementação em hardware do sistema. Neste caso, pode-se observar que a comunicação é unidirecional entre os P.E.s e bidirecional entre cada P.E e os outros elementos do sistema.

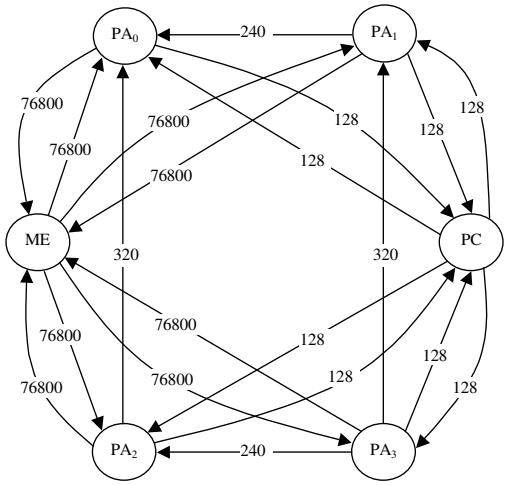


Figura 3: Grafo de comunicação do exemplo de segmentação de imagens.

APÊNDICE I – CÓDIGO FONTE DOS COMPONENTES

Elemento de Processamento

```
//Modificações no número de entradas e saídas afetam estas definições, os pinos
// de entrada, pinos de saída e a rotina de escrita nas saídas.
#define NUMMAXREGIOES 256 //numero maximo de regioes a serem computadas as areas

//consideracoes:
//0=vazio
//255=cheio
//outro valor=numero da regiao a qual pertence
enum {LESTE, NORTE, OESTE, SUL, FOI};
enum {RECMEMORIA, RECCOLUNA, RECLINHA, RECREGIOES, RECNUMERACAO, ENVMEMORIA};

class ProcCentral;

class ProcElem{
  public:
  unsigned int linha_memoria;
  unsigned int pos_x;
  unsigned int pos_y;
  unsigned int valor_anterior;
  unsigned int area;
  unsigned int areas_locais[NUMMAXREGIOES];//deve poder conter TAMX*TAMY
  unsigned char numero_do_processador;
  unsigned char idt_regiao;
  unsigned int numero_de_regioes;//tipo depende do numero maximo de regioes
  unsigned int numero_real_de_regioes;//tipo depende do numero maximo de regioes
  unsigned int * numeracao_de_saida;

  //--------------------
  unsigned char conteudo[TAMY][TAMX];//deve conter a imagem de entrada
  unsigned char conteudoM[TAMY][TAMX];//deve conter a imagem rotulada como saida
  unsigned char linha_superior_vizinho[TAMX];
  unsigned char linha_superior_vizinhoM[TAMX];
  unsigned char linha_superior[2*TAMX];//Armazena intensidades(0~TAMX-1) e regioes(TAMX~2TAMX-1)
  unsigned char coluna_esquerda_vizinho[TAMY];
  unsigned char coluna_esquerda_vizinhoM[TAMY];
  unsigned char coluna_esquerda[2*TAMY];//Armazena intensidades(0~TAMY-1) e regioes(TAMY~2TAMY-1)
  unsigned int linha_memoria_saida;
  bool linha_recebida;
  bool coluna_recebida;

  bool conteudo_visitado[TAMY][TAMX];
  bool direcao[TAMY][TAMX];

  //--------------------
  class pilha{
    public:
    unsigned int pilhaCX[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int pilhaCY[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int pilhaVA[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int num_elem;
    pilha(){
      num_elem=0;
      }
    void EmpilhaCoordenadas(unsigned int x,unsigned int y,unsigned int va){
      pilhaCX[num_elem]=x;
      pilhaCY[num_elem]=y;
      pilhaVA[num_elem]=va;
      num_elem++;
      }
    void DesempilhaCoordenadas(unsigned int * x,unsigned int * y,unsigned int * va){
      num_elem--;
      (*x)=pilhaCX[num_elem];
      (*y)=pilhaCY[num_elem];
      (*va)=pilhaVA[num_elem];
      }
    }Pilha;
  //--------------------
  class pilhamod:pilha{
```

```
    public:
    void Adiciona(unsigned int a,unsigned int b){
      bool encontrou=false;
      for(unsigned int i=0;i<num_elem;i++)
        if((pilhaCX[i]==a)&&(pilhaCY[i]==b))encontrou=true;
      if(encontrou==false)EmpilhaCoordenadas(a,b,0);
      }
    unsigned int NumeroDeElementos(){
      return num_elem;
      }
    void CopiaRP(unsigned int * rp){//regiao primaria(passa a numeracao)
      for(unsigned int i=0;i<num_elem;i++)
        rp[i]=pilhaCX[i];
      return;
      }
    void CopiaRS(unsigned int * rs){//regiao secundaria(recebe a numeracao)
      for(unsigned int i=0;i<num_elem;i++)
        rs[i]=pilhaCY[i];
      return;
      }
    }InterseccoesH,InterseccoesV;
  //--------------------
  ProcElem()
    {
    Reset();
    }

  ~ProcElem(){
    if(numeracao_de_saida!=NULL)delete [] numeracao_de_saida;
    }

  void Reset();
  void Executa();
  void VisitaElemento();
  void PegaIdentificacao(unsigned char num);
  void RecebeMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned char tipo);
  void RecebeMensagemW(unsigned int * conteudo_msg, unsigned int comprimento,unsigned char tipo);
  void EnviaMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned char tipo);
  void ProcessaFronteiras();
  void EnviaFronteiras();


  ProcElem * PEEsq;
  ProcElem * PESup;
  ProcCentral * PC;

};

#include "systemc.h"

#include "geral.h"
#include "ProcCentral.h"
#include "ProcElem.h"

//------------------------------------------------------------------------------
void ProcElem::Executa()
{
  numero_de_regioes=0;//tipo depende do numero maximo de regioes
  valor_anterior=1;
  idt_regiao=1;
  for(unsigned int posicao_y=0;posicao_y<TAMY;posicao_y++){
    for(unsigned int posicao_x=0;posicao_x<TAMX;posicao_x++){
      pos_x=posicao_x;
      pos_y=posicao_y;
      area=0;
      valor_anterior=conteudo[pos_y][pos_x];
      VisitaElemento();
      if(area!=0){
        numero_de_regioes++;
        idt_regiao++;
        }
      }
    }
  printf("PE no.%d encontrou %d ",numero_do_processador,numero_de_regioes);
  if(numero_de_regioes!=1)
    printf("regioes.\n");
  else
```

```
    printf("regiao.\n");

  //Comunica com vizinho superior
  bool toca_borda=false;
  for(int x=0;x<TAMX;x++){
    linha_superior[x]=conteudo[0][x];
    linha_superior[TAMX+x]=conteudoM[0][x];
    if(linha_superior[TAMX+x]!=0)toca_borda=true;
    }

  if(toca_borda==true){
    //envia mensagem
    if(PESup!=NULL){
      PESup->RecebeMensagem(linha_superior,2*TAMX,RECLINHA);
      }
    }

  //Comunica com o vizinho da esquerda
  toca_borda=false;
  for(int y=0;y<TAMY;y++){
    coluna_esquerda[y]=conteudo[y][0];
    coluna_esquerda[TAMY+y]=conteudoM[y][0];
    if(coluna_esquerda[TAMY+y]!=0)toca_borda=true;
    }

  if(toca_borda==true){
    //envia mensagem
    if(PEEsq!=NULL){
      PEEsq->RecebeMensagem(coluna_esquerda,2*TAMY,RECCOLUNA);
      }
    }
 }
//-------------------------------------------------------------------------------
void ProcElem::VisitaElemento(){
    if((conteudo_visitado[pos_y][pos_x]==false)&&(conteudo[pos_y][pos_x]!=0)){
      conteudo_visitado[pos_y][pos_x]=true;//deve marcar aqui para os subniveis nao repercorrerem o
caminho
      conteudoM[pos_y][pos_x]=idt_regiao;
      if(pos_x==159)
        pos_x=pos_x;
      area++;
      if(pos_x>0){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_x-=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_y>0){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_y-=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_x<TAMX-1){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_x+=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_y<TAMY-1){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_y+=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      }
    }
//-------------------------------------------------------------------------------
void ProcElem::PegaIdentificacao(unsigned char numero){
  numero_do_processador=numero;
  }
//-------------------------------------------------------------------------------
void ProcElem::RecebeMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned char
tipo){
  if(tipo==RECMEMORIA){
    for(unsigned int x=0;x<comprimento;x++)
      conteudo[linha_memoria][x]=conteudo_msg[x];
```

```
    linha_memoria++;
    }
  if(tipo==RECLINHA){
    for(unsigned int x=0;x<comprimento/2;x++){
      linha_superior_vizinho[x]=conteudo_msg[x];
      linha_superior_vizinhoM[x]=conteudo_msg[x+comprimento/2];
      }
    linha_recebida=true;
    printf("Recebida linha no processador %d\n",numero_do_processador);
    }
  if(tipo==RECCOLUNA){
    for(unsigned int y=0;y<comprimento/2;y++){
      coluna_esquerda_vizinho[y]=conteudo_msg[y];
      coluna_esquerda_vizinhoM[y]=conteudo_msg[y+comprimento/2];
      }
    coluna_recebida=true;
    printf("Recebida coluna no processador %d\n",numero_do_processador);
    }
  }
//-------------------------------------------------------------------------------
void ProcElem::RecebeMensagemW(unsigned int * conteudo_msg, unsigned int comprimento,unsigned char
tipo){
  if(tipo==RECNUMERACAO){
    if(numeracao_de_saida!=NULL)delete [] numeracao_de_saida;
    numeracao_de_saida=new(unsigned int [comprimento+1] );
    numeracao_de_saida[0]=0;
    for(unsigned int i=1;i<=comprimento;i++){
      numeracao_de_saida[i]=conteudo_msg[i];
      }
    }
  printf("Recebida a numeracao de saida do processador %d\n",numero_do_processador);
  numero_real_de_regioes=conteudo_msg[comprimento+1];
  printf("Numero de regioes corrigido para %d\n",conteudo_msg[comprimento+1]);
  extern unsigned int total;
  total+=conteudo_msg[comprimento+1];
  }
//-------------------------------------------------------------------------------
void ProcElem::EnviaMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned char
tipo){
  if(tipo==ENVMEMORIA){
    for(unsigned int x=0;x<comprimento;x++)
      conteudo_msg[x]=(unsigned char)numeracao_de_saida[conteudoM[linha_memoria_saida][x]];
    linha_memoria_saida++;
//    numero_real_de_regioes=conteudo_msg[comprimento];
    }
  }
//-------------------------------------------------------------------------------
void ProcElem::Reset(){
  pos_x=0;
  pos_y=0;
  area=0;
  linha_memoria=0;
  linha_memoria_saida=0;
  linha_recebida=false;
  coluna_recebida=false;
  idt_regiao=1;
  PEEsq=NULL;
  PESup=NULL;
  numero_de_regioes=0;
  for(int x=0;x<TAMX;x++){
    linha_superior_vizinho[x]=0;
    linha_superior_vizinhoM[x]=0;
    linha_superior[x]=0;
    linha_superior[x+TAMX]=0;
    }
  for(int y=0;y<TAMY;y++){
    coluna_esquerda_vizinho[y]=0;
    coluna_esquerda_vizinhoM[y]=0;
    coluna_esquerda[y]=0;
    coluna_esquerda[y+TAMY]=0;
    }
  for(int y=0;y<TAMY;y++){
    for(int x=0;x<TAMX;x++){
      conteudo[y][x]=0;
      conteudoM[y][x]=0;
      conteudo_visitado[y][x]=false;//marca como nao visitado
      }
```

```
    }
  numeracao_de_saida=NULL;
  numero_real_de_regioes=0;
  valor_anterior=0;
  }
//-------------------------------------------------------------------------------
void ProcElem::ProcessaFronteiras(){
  //Compara com base em 4 vizinhos
  printf("Processador %d comparando fronteiras\n");
  //Linha
  for(int x=0;x<TAMX;x++){
    if(abs(conteudo[TAMY-1][x]-linha_superior_vizinho[x])<TOLERANCIA)
      if((conteudoM[TAMY-1][x]==0)||(linha_superior_vizinhoM[x]==0))
        x=x;
      else
        InterseccoesV.Adiciona(conteudoM[TAMY-1][x],linha_superior_vizinhoM[x]);
    }
  //Coluna
  for(int y=0;y<TAMY;y++){
    if(abs(conteudo[y][TAMX-1]-coluna_esquerda_vizinho[y])<TOLERANCIA)
      if((conteudoM[y][TAMX-1]==0)||(coluna_esquerda_vizinhoM[y]==0))
        y=y;
      else
        InterseccoesH.Adiciona(conteudoM[y][TAMX-1],coluna_esquerda_vizinhoM[y]);
    }
  printf("Processador                    %d                    encontrou                    %d
intersecções.\n",numero_do_processador,InterseccoesH.NumeroDeElementos()+InterseccoesV.NumeroDeEleme
ntos());
  }
//-------------------------------------------------------------------------------
void ProcElem::EnviaFronteiras(){
  unsigned int * RPH,* RSH,* RPV,* RSV;
  RPH=new(unsigned int[InterseccoesH.NumeroDeElementos()]);
  RSH=new(unsigned int[InterseccoesH.NumeroDeElementos()]);
  RPV=new(unsigned int[InterseccoesV.NumeroDeElementos()]);
  RSV=new(unsigned int[InterseccoesV.NumeroDeElementos()]);
  InterseccoesH.CopiaRP(RPH);
  InterseccoesH.CopiaRS(RSH);
  InterseccoesV.CopiaRP(RPV);
  InterseccoesV.CopiaRS(RSV);
  PC-
>RecebeMensagem(RPH,RSH,InterseccoesH.NumeroDeElementos(),RPV,RSV,InterseccoesV.NumeroDeElementos(),
    numero_de_regioes,numero_do_processador);
  delete [] RPH;
  delete [] RSH;
  delete [] RPV;
  delete [] RSV;
  }
//-------------------------------------------------------------------------------
```

Processador Central

```
//Modificações no número de entradas e saídas afetam estas definições, os pinos
// de entrada, pinos de saída e a rotina de escrita nas saídas.
#define NUMMAXREGIOES 1024 //numero maximo de regioes a serem computadas as areas

//consideracoes:
//0=vazio
//255=cheio
//outro valor=numero da regiao a qual pertence

class ProcElem;

class ProcCentral{
  class PE{
    public:
    unsigned int * pxH;
    unsigned int * pyH;
    unsigned int * pxV;
    unsigned int * pyV;
    unsigned int neph;
    unsigned int nepv;
    unsigned int netotal;
    unsigned int *regioes_old;
```

```
    unsigned int **regioes_new;
    unsigned char n_regioes;
      unsigned int menor;
      unsigned int indice;
      unsigned int dependencias[255];
    PE(){
      regioes_old=NULL;
      regioes_new=NULL;
      pxH=NULL;    pyH=NULL;    pxV=NULL;    pyV=NULL;
      neph=0;      nepv=0;      netotal=0;
      n_regioes=0;
      }
    ~PE(){
      if(pxH!=NULL)delete[]pxH;
      if(pyH!=NULL)delete[]pyH;
      if(pxV!=NULL)delete[]pxV;
      if(pyV!=NULL)delete[]pyV;
      if(regioes_old!=NULL)delete[]regioes_old;
      for(int i=0;i<netotal;i++)
        if(regioes_new[i]!=NULL)delete[]regioes_new[i];
      if(regioes_new!=NULL)delete[]regioes_new;
      }
    //Aloca espaço para a tabela de renomeação das regiões e outras informações
    void Aloca(unsigned int h,unsigned int v,unsigned int total){
      pxH=new(unsigned int [h]);
      pyH=new(unsigned int [h]);
      pxV=new(unsigned int [v]);
      pyV=new(unsigned int [v]);
      netotal=total;
      neph=h;
      nepv=v;
      regioes_old=new(unsigned int [netotal]);
      regioes_new=new(unsigned int * [netotal]);
      for(int i=0;i<netotal;i++)
        regioes_new[i]=NULL;
      }
    void MontaTabela(unsigned int * contador){
      for(unsigned int posicao=0;posicao<netotal;posicao++){
        regioes_new[posicao]=new(unsigned int [neph+nepv+1]);
        for(unsigned int i=0;i<neph+nepv+1;i++)
          regioes_new[posicao][i]=0;
        regioes_old[posicao]=posicao+1;
        regioes_new[posicao][0]=(* contador);
        printf("Regioes_new[%d]=%d\n",posicao,regioes_new[posicao][0]);
        (* contador)++;
        }
      }
    void AdicionaDependenciasH(PE * PEE){
      for(unsigned int dependencia_h=0;dependencia_h<neph;dependencia_h++){
        unsigned int pos=1;
        while(regioes_new[pxH[dependencia_h]-1][pos]!=0)pos++;
        regioes_new[pxH[dependencia_h]-1][pos]=PEE->regioes_new[pyH[dependencia_h]-1][0];
        printf("Regioes_newH[%d][%d]=%d\n",pxH[dependencia_h]-1,pos,regioes_new[pxH[dependencia_h]-
1][pos]);
        }
      }
    void AdicionaDependenciasV(PE * PEE){
      for(unsigned int dependencia_v=0;dependencia_v<nepv;dependencia_v++){
        unsigned int pos=1;
        while(regioes_new[pxV[dependencia_v]-1][pos]!=0)pos++;
        regioes_new[pxV[dependencia_v]-1][pos]=PEE->regioes_new[pyV[dependencia_v]-1][0];
        printf("Regioes_newV[%d][%d]=%d\n",pxV[dependencia_v]-1,pos,regioes_new[pxV[dependencia_v]-
1][pos]);
        }
      }
    void SubstituiDependencias(unsigned int *lista_de_dependencias,unsigned int n, unsigned int
novo_valor){
      for(unsigned int linha=0;linha<netotal;linha++){
        for(unsigned int pos_linha=0;pos_linha<neph+nepv+1;pos_linha++){
          if(regioes_new[linha][pos_linha]==0)break;
          for(unsigned int pos_lista=0;pos_lista<n;pos_lista++){
            if(regioes_new[linha][pos_linha]==lista_de_dependencias[pos_lista])
              regioes_new[linha][pos_linha]=novo_valor;
            }
          }
        }
      }
```

```
    void ProcessaDependencias(unsigned int posicao){
        menor=32767;
        for(indice=0;indice<neph+nepv+1;indice++){
          dependencias[indice]=regioes_new[posicao][indice];
          if(regioes_new[posicao][indice]==0)break;
          if(regioes_new[posicao][indice]<menor)menor=regioes_new[posicao][indice];
          if(indice>0)regioes_new[posicao][indice]=0;
          }
        if((menor!=32767)&&(indice!=1)){
          printf("Menor:%d\n",menor);
          }
      }
    void Copia(unsigned int * pCXH, unsigned int * pCYH,unsigned int * pCXV, unsigned int * pCYV){
      for(unsigned int i=0;i<neph;i++){
        pxH[i]=pCXH[i];
        pyH[i]=pCYH[i];
        printf("(%d,%d)h\n",pxH[i],pyH[i]);
        }
      for(unsigned int i=0;i<nepv;i++){
        pxV[i]=pCXV[i];
        pyV[i]=pCYV[i];
        printf("(%d,%d)v\n",pxV[i],pyV[i]);
        }
      }
    void  MontaTabelaDeSaida(unsigned  int  *  TabelaDeSaida,unsigned  int  *  tabela,  unsigned  int
*contador){
      for(unsigned int posicao=0;posicao<netotal;posicao++){
        if(TabelaDeSaida[regioes_new[posicao][0]]==0){
          TabelaDeSaida[regioes_new[posicao][0]]=(* contador);

printf("TabelaDeSaida[%d]=%d\n",regioes_new[posicao][0],TabelaDeSaida[regioes_new[posicao][0]]);
          (*contador)++;
          n_regioes++;
          }
        }
      tabela[0]=0;
      for(unsigned int i=0;i<netotal;i++){
        tabela[i+1]=TabelaDeSaida[regioes_new[i][0]];
        printf("Tabela[%d]=%d\n",i+1,tabela[i+1]);
        }
      }
    }PELocal[NPESY][NPESX];//PELocal1,PELocal2,PELocal3,PELocal4;

  public:
  ProcCentral(){
    Reset();
    }

  ~ProcCentral(){
    }

  void Reset();
  void RecebeMensagem(
    unsigned int * pCXH, unsigned int * pCYH, unsigned int num_elem_h,
    unsigned int * pCXV, unsigned int * pCYV,unsigned int num_elem_v,
    unsigned int total,int num_pe);
  void ProcessaRegioes();
  ProcElem * PEReal[NPESY][NPESX];
  bool PEEnviou[NPESY][NPESX];

};

#include "geral.h"
#include "ProcElem.h"
#include "ProcCentral.h"

//---------------------------------------------------------------------------
void ProcCentral::RecebeMensagem(
  unsigned int * pCXH, unsigned int * pCYH, unsigned int num_elem_h,
  unsigned int * pCXV, unsigned int * pCYV,unsigned int num_elem_v,
  unsigned int total,int num_pe){

  //Recebe dados do elemento de processamento num_pe
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      if(PEReal[y][x]->numero_do_processador==num_pe){
        PELocal[y][x].Aloca(num_elem_h,num_elem_v,total);
```

```
        PELocal[y][x].Copia(pCXH,pCYH,pCXV,pCYV);
        }
  printf("Recebida mensagem do PE%d\n",num_pe);
  }
//---------------------------------------------------------------------------
void ProcCentral::Reset(){
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      PEEnviou[y][x]=false;
      PEReal[y][x]=NULL;
      }
  }
//---------------------------------------------------------------------------
void ProcCentral::ProcessaRegioes(){
  unsigned int contador=1;
  printf("Montando tabela\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      PELocal[y][x].MontaTabela(&contador);
  printf("Adicionando dependencias\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX-1;x++)
      PELocal[y][x].AdicionaDependenciasH(&PELocal[y][x+1]);
  for(int y=0;y<NPESY-1;y++)
    for(int x=0;x<NPESX;x++)
      PELocal[y][x].AdicionaDependenciasV(&PELocal[y+1][x]);
  printf("Processando e substituindo dependencias\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      for(unsigned int posicao=0;posicao<PELocal[y][x].netotal;posicao++){

      PELocal[y][x].ProcessaDependencias(posicao);
      for(int yp=0;yp<NPESY;yp++)
        for(int xp=0;xp<NPESX;xp++)

PELocal[yp][xp].SubstituiDependencias(PELocal[y][x].dependencias,PELocal[y][x].indice,PELocal[y][x].
menor);
        }
      }
  printf("Montando tabelas de saida\n");
  unsigned int * TabelaDeSaida;
  unsigned int tamanho=1;//necessario
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      tamanho+=PELocal[y][x].netotal;
  TabelaDeSaida=new(unsigned int [tamanho]);

  for(unsigned int i=0;i<tamanho;i++)
    TabelaDeSaida[i]=0;

  unsigned int * TabelaDoProcessador;
  contador=1;

  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      TabelaDoProcessador=new(unsigned int [PELocal[y][x].netotal+2]);
      PELocal[y][x].MontaTabelaDeSaida(TabelaDeSaida,TabelaDoProcessador,&contador);
      TabelaDoProcessador[PELocal[y][x].netotal+1]=PELocal[y][x].n_regioes;//ultimo  elemento=numero
real de regioes
      PEReal[y][x]->RecebeMensagemW(TabelaDoProcessador,PELocal[y][x].netotal,RECNUMERACAO);
      delete [] TabelaDoProcessador;
      }

  //devolve numeracao para processadores
  delete[]TabelaDeSaida;
  }
//---------------------------------------------------------------------------
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Em andamento...