Uso de Paralelismo de Dados em Algoritmos de Processamento de Imagens Utilizando Haskell

Pedro R. Mendes Júnior
Lucília Figueiredo
David Menotti

Departamento de Computação
Universidade Federal de Ouro Preto

26 de outubro de 2012

Organização

Introdução

Revisão da Literatura

Desenvolvimento

- Cálculo da Árvore de Componentes Conectados
- Data Parallel Haskell
- Concurrent Haskell

Experimentos

Conclusões

Introdução

Introdução

Objetivos:

- Capacidade processamento paralelo;
- Linguagem funcional (Haskell); Data Parallel Haskell (DPH); Concurrent Haskell (CH);
- Algoritmos de Processamento Digital de Imagens (PDI); [Matas et al., 2008];
- Eficiência das implementações; comparações; implementações em outras linguagens;

Introdução

Formas de paralelismo:

- Paralelismo de controle; criação explícita de *threads*; sincronização por meio de mensagens e *locks*;
- Paralelismo de dados; conjunto de instruções em um conjunto de dados; emprego de um grande número de processadores.

Introdução

Trabalho de Blelloch em NESL (*NESted-parallel Language*):

- Paralelismo de Dados Aninhados (NDP – *Nested Data Parallelism*); modelo flexível;
- Paralelismo Plano de Dados (FDP – *Flat Data Parallelism*); execução eficiente; grande escalabilidade;
- Data Parallel Haskell (DPH); extensão do compilador GHC.

Introdução

Então,

- Área de PDI parece ser fértil; existem exemplos de sucesso; *smoothing*; segmentação; abordagens tradicionais;
- Duas das principais extensões Haskell; DPH; CH;
- Algoritmos não tradicionais de PDI; saída não corresponde a uma imagem;
- Árvore de Componentes Conectados (CCT – *Connected Component Tree*); algoritmo de [Matas et al., 2008].

Revisão da Literatura

Revisão da Literatura

Paralelismo e concorrência:

- Programação paralela; modelo determinístico; aproveitar os recursos; eficiência; facilidade de realizar testes; código puro; paralelismo de dados;
- Programação concorrente; modelo não determinístico; estrutura do programa; interagir com agentes externos; paralelismo de controle.
- Grande diferenciação entre programação paralela e programação concorrente; diferenciação nem tão evidenciada em outras linguagens;

Observação: o modelo de programação determinístico não é suficiente para expressar todos os tipos de algoritmos paralelos [Marlow, 2012].

Revisão da Literatura

Data Parallel Haskell (DPH):

- Biblioteca que mais nos chamou a atenção; modelo de NDP; sintaxe semelhante à sintaxe de códigos sequenciais;
- Paralelismo com base no *array* paralelo; substitui o tipo `List` de Haskell; pequenas diferenças na notação.

Revisão da Literatura

Diferenças de notação com o tipo `List` de Haskell:

- Denotação do *array* paralelo; lista de elementos do tipo `e`: `[e]`; *array* paralelo de elementos do tipo `e`: `[:e:]`;
- Funções de ordem superior; adiciona-se o sufixo `P`; `map` → `mapP`; `filter` → `filterP`; `unzip` → `unzipP`;
- *Array comprehension*;
  list comprehension:

```
[x*y | x <- list, y <- [1, 2, 3]]
```

  parallel array comprehension:

```
[:x*y | x <- parray, y <- [:1, 2, 3:]:]
```

Revisão da Literatura

Concurrent Haskell (CH):

- Programação com *threads* explícitas; mônada `MVar` para abstrair a comunicação de baixo nível; operações atômicas; quatro primitivas que permitem seu uso;

```
newEmptyMVar :: IO (MVar a)
takeMVar :: MVar a -> IO a
putMVar :: MVar a -> a -> IO ()
forkIO :: IO () -> IO ThreadId
```

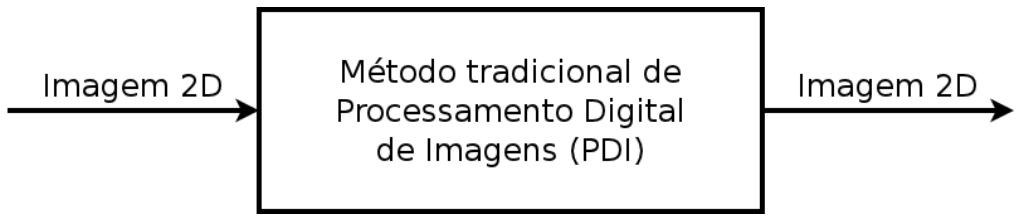
- Utilizamos a estrutura `MVector`; estrutura não persistente; representação da *point-tree*.

Desenvolvimento

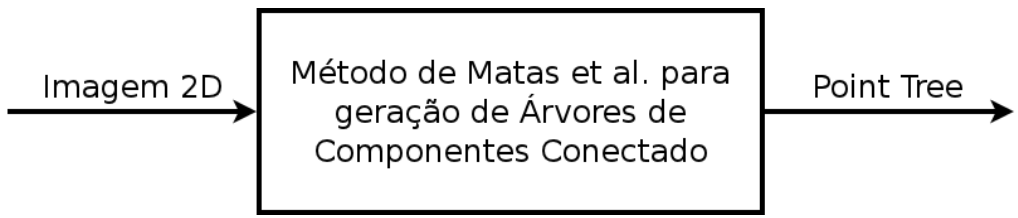
Desenvolvimento

Objetivos durante o desenvolvimento:

- Estudar a capacidade de paralelismo em Haskell; avaliar vantagens e desvantagens; encontrar pontos para possível desenvolvimento;
- Algoritmos de PDI; cálculo da CCT; algoritmo não tradicional.



(a)



(b)

Cálculo da Árvore de Componentes Conectados

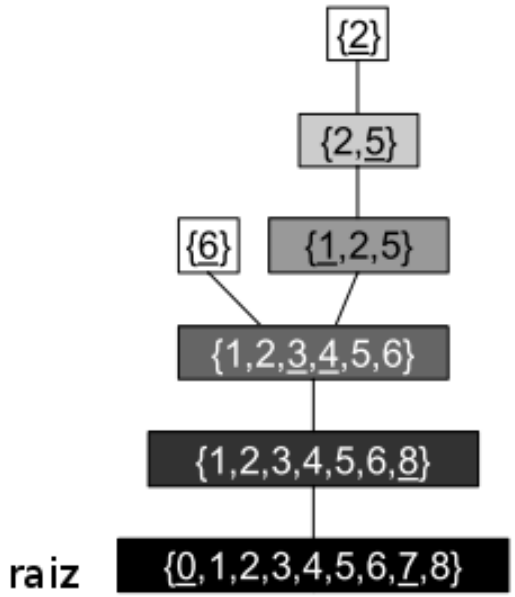
Cálculo da Árvore de Componentes Conectados

Representação da CCT:

- Matriz *point-tree*; mesma dimensão que a imagem; os elementos são as coordenadas do pai; raiz possui coordenada não válida;
- Representação apropriada; verificação e atualização $O(1)$; evitamos a utilização de árvores;
- Cálculos parciais sobre uma dimensão; paralelamente; junções dos resultados parciais; vários níveis de junções; cada nível possui junções paralelas.

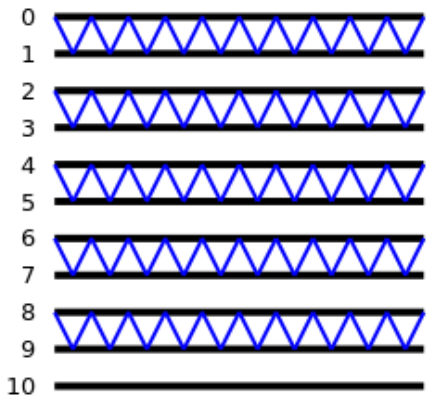
Cálculo da Árvore de Componentes Conectados

(c)

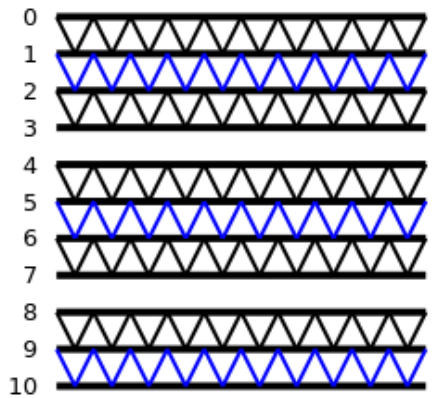


(d)

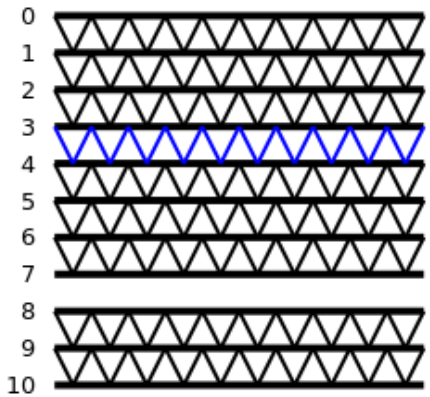
Cálculo da Árvore de Componentes Conectados



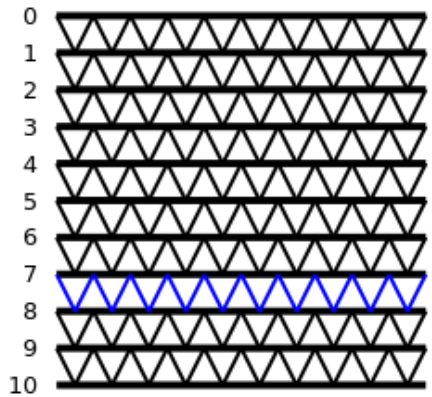
(e)



(f)



(g)



(h)

Data Parallel Haskell

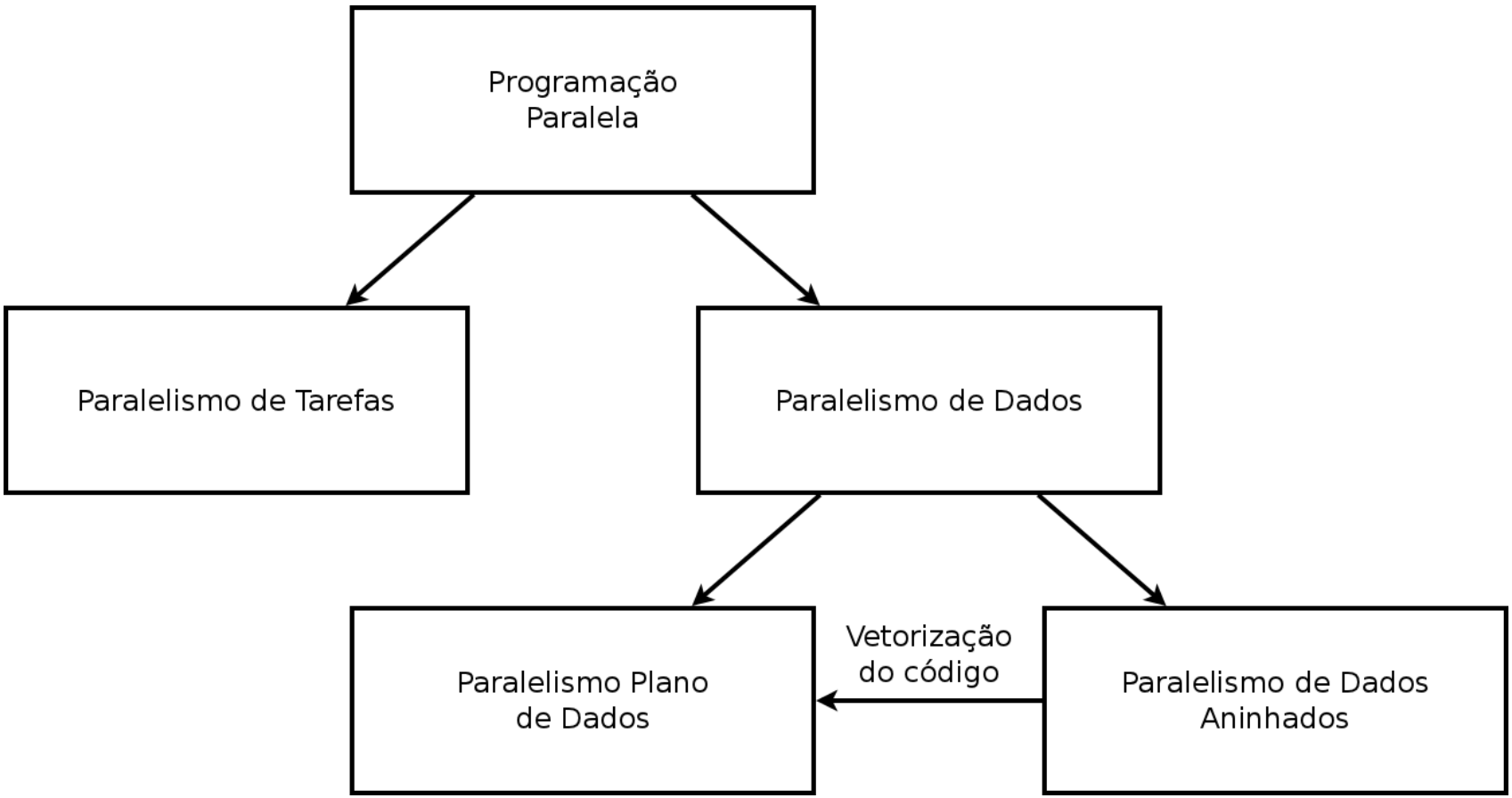
Data Parallel Haskell

Características:

- Extensão que mais nos chamou a atenção; NDP; área de PDI aparentou ser fértil; representação da imagem como *array* de *arrays*;
- Código semelhante ao código sequencial; paralelismo automático; *arrays* paralelos.

Revisão da Literatura

Etapa de "vetorização" [Jones et al., 2008]:



Data Parallel Haskell

Limitações:

- API limitada; diferente da API sobre listas; sem as funções `foldl` e `foldr`; etapas sequenciais não podem ser realizadas "por dentro" do DPH;
- Impossibilidade de uso de código não "vetorizável"; exemplo do *prelude* próprio; etapas sequenciais não podem ser realizadas "por fora" do DPH.

Observação: A etapa de cálculo da CCT sobre uma dimensão não é possível de ser paralelizada.

Concurrent Haskell

Concurrent Haskell

Características:

- Concorrência; sem garantia de determinismo; uso de uma mônada de estados;
- Apropriado a utilização de `MVector`; alterações não sequenciais na *point-tree*; dependentes do fluxo de execução.

Concurrent Haskell

Implementação:

- Semelhante a uma implementação imperativa; objetivamos a simples comparação de tempo;
- Semelhante a uma implementação sequencial; definição da função `mapM_P`; mesmo tipo que a função `mapM_`.

Concurrent Haskell

```
mapM_P :: Monad m => (a -> m b) -> [a] -> m ()
mapM_P f xs = do
  mvars <- myReplicate (length xs) newEmptyMVar
  mapM_ (forkIO . fThread) $ zip mvars xs
  mapM_ takeMVar mvars
  where fThread (mvar, l) = f l >> putMVar mvar ()
        myReplicate :: Int -> IO a -> IO [a]
        myReplicate n x = sequence $ replicate n x
```

```
mapM_P matas1D [0 .. height - 1]
```

Resultados e Experimentos

Resultados e Experimentos

Implementações do algoritmo de [Matas et al., 2008]:

- Haskell; Concurrent Haskell;
- C++; PThreads;
- MATLAB; Parallel Computing Toolbox.

Resultados e Experimentos

- 7 imagens; 10 execuções em cada imagem;
- Intel Pentium i3; 4 processadores.

	Haskell	MATLAB
Seq.	7,72 ± 0,11	7200
1 *thread*	7,71 ± 0,15	7200
2 *threads*	5,14 ± 0,23	7200
4 *threads*	5,12 ± 0,19	7200

	C++
Seq.	2,17 ± 0,06
Par.	0,83 ± 0,02

- Haskell *speedup*:

$$\frac{7,72}{5,12} = 1,51$$

- C++ *speedup*:

$$\frac{2,17}{0,83} = 2,61$$

Conclusões

Conclusões

- Paralelismo em Haskell; Data Parallel Haskell; Concurrent Haskell; Árvore de Componentes Conectados;
- Insucesso com o DPH; limitação na API; impossibilidade de mesclar código não paralelo;
- Avaliação com o CH; semelhante a uma linguagem imperativa; perda da "pureza"; utilização de `MVector`;
- Comparações; Haskell (CH); C++ (PThreads); MATLAB (inviável);
- Matrizes em Haskell; Haskell persistente; atualizações em matrizes;
- Trabalho futuro; possibilidade de incremento na API do DPH.

Obrigado!


Jones, S. P., Leshchinskiy, R., Keller, G., and Chakravarty, M. M. T. (2008).
Harnessing the multicores: Nested data parallelism in haskell.
In Hariharan, R., Mukund, M., and Vinay, V., editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 383–414, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.


Marlow, S. (2012).
Parallel and concurrent programming in haskell.
In Zsók, V., Horváth, Z., and Plasmeijer, R., editors, *Central European Functional Programming School*, volume 7241 of *Lecture Notes in Computer Science*, pages 339–401. Springer Berlin / Heidelberg.
10.1007/978-3-642-32096-5_7.


Matas, P., Dokládalová, E., Akil, M., Grandpierre, T., Najman, L., Poupa, M., and Georgiev, V. (2008).

Parallel algorithm for concurrent computation of connected component tree.
In Blanc-Talon, J., Bourennane, S., Philips, W., Popescu, D., and Scheunders, P., editors, *Advanced Concepts for Intelligent Vision Systems*, volume 5259 of *Lecture Notes in Computer Science*, pages 230–241. Springer Berlin / Heidelberg. 10.1007/978-3-540-88458-3_21.