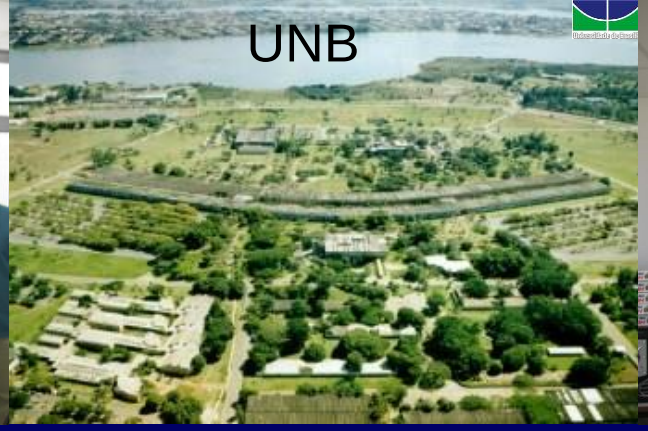


Programação Paralela em Plataformas de Alto Desempenho: Estudos de Caso em Bioinformática

Alba Cristina Magalhaes Alves de Melo

Professor Titular na Universidade de Brasília, UNB, Brasil

Bolsista de Produtividade e Pesquisa PQ 1D – CNPq

ERAD-SP, 8 de abril de 2017

Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação de Sequências Biológicas
- Tarefas independentes: Package-BLAST
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusões

Introdução

- Programação paralela é um paradigma de programação onde uma aplicação é quebrada em diversas entidades (processos, tarefas, threads, etc) que podem ser executadas simultaneamente.
- Esse paradigma existe desde a década de 1960 e, apesar da evolução impressionante dos computadores, suas bases continuam as mesmas.
- O objetivo da programação paralela é obter resultados corretos de maneira rápida.

Introdução

- Na era dos multicores e aceleradores, a programação paralela tem um papel fundamental no sucesso de um projeto.
- Mesmo assim, uma parcela relativamente pequena dos programadores estuda essa importante área e, consequentemente, são produzidos programas de baixíssima qualidade.
  - Produzem resultados errados e/ou
  - Demoram muito tempo
- Dentre outras, a área de Bioinformática possui muitos problemas que podem tirar alto proveito da programação paralela.

Objetivo da Palestra

- Discorrer sobre os príncipios da programação paralela e apresentar estudos de caso em Bioinformática onde a aplicação paralela é composta:
  - Tarefas independentes - Grid
  - Tarefas dependentes
    - Placas gráficas – GPU
    - Hardware reconfigurável – FPGA

Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação Pairwise de Sequências Biológicas
- Tarefas independentes: PackageBLAST
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusão

Tipos de Aplicação Paralela

- A aplicação paralela é composta de múltiplas tarefas, que podem ser:
  - Independentes
    - *Bag of Tasks – BoT*
  - Dependentes
    - de entrada e saída (*workflow)*
    - na mesma etapa (*communicating tasks)*

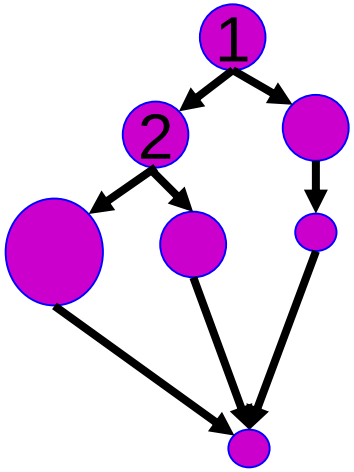
Tipos de Aplicação Paralela

Bag-of-Tasks

as tarefas podem ser executadas em qualquer ordem

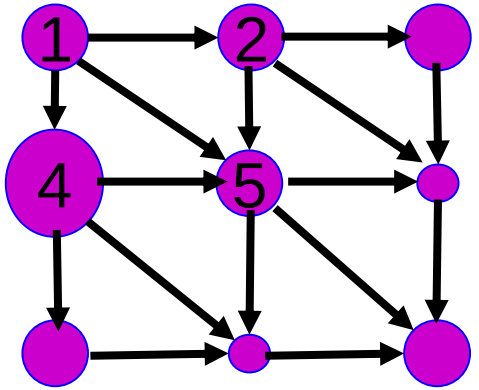
Workflow

o input da tarefa 2 é o output da tarefa 1



Communicating Tasks

As tarefas possuem um fluxo de comunicação frequente



tarefa

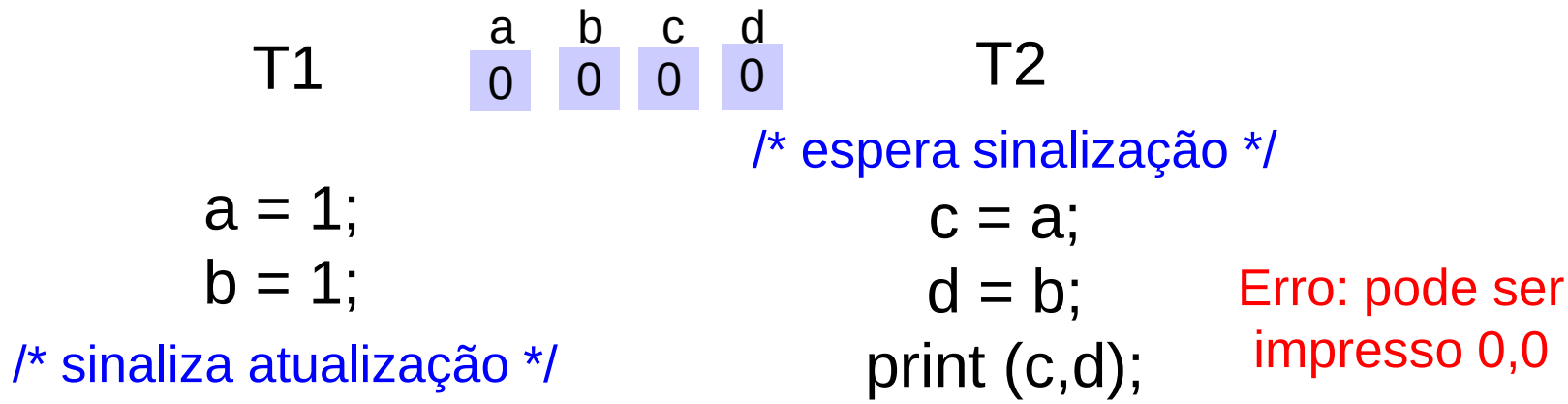
dependência (comunicação)

Aplicação BoT

- Como as tarefas são independentes, o problema a ser enfrentado é garantir que a execução de todas as tarefas se complete rápido
  - Mapeamento entre tarefas e unidades de processamento deve considerar balanceamento de carga

Aplicações com Dependência

- Deve ser garantido que as tarefas dependentes não usem dados que ainda não foram produzidos:
  - Condição de corrida e sincronização



Aplicação com Dependência

- Deve ser garantido que a aplicação não vai ficar bloqueada eternamente em alguma execução.

a	b	c	d
0	0	0	0

T1	T2
	/* espera sinalização */
a = 1;	c = a;
b = 1;	d = b;
	print (c,d);

Deadlock!!!

Programação Paralela

- Um bom programa paralelo vai se executar mais rápido que a versão sequencial, geralmente sem condição de corrida e com certeza sem deadlock.
  - A sincronização faz com que o programa se execute mais lento

Programação Paralela

- O que fazer se o programa é estritamente sequencial?
  - Executar em uma única tarefa ou
  - Especulação

T1

a	b	c	d	…
0	0	0	0	

```
b = rot1(a);
c = rot2(b);
d =rot3(c);
e=rot4(d);
res = calcula(e);
print(res);
/* comunica o valor de e para T2 */
```

T2

Observo que na maioria das vezes e=1

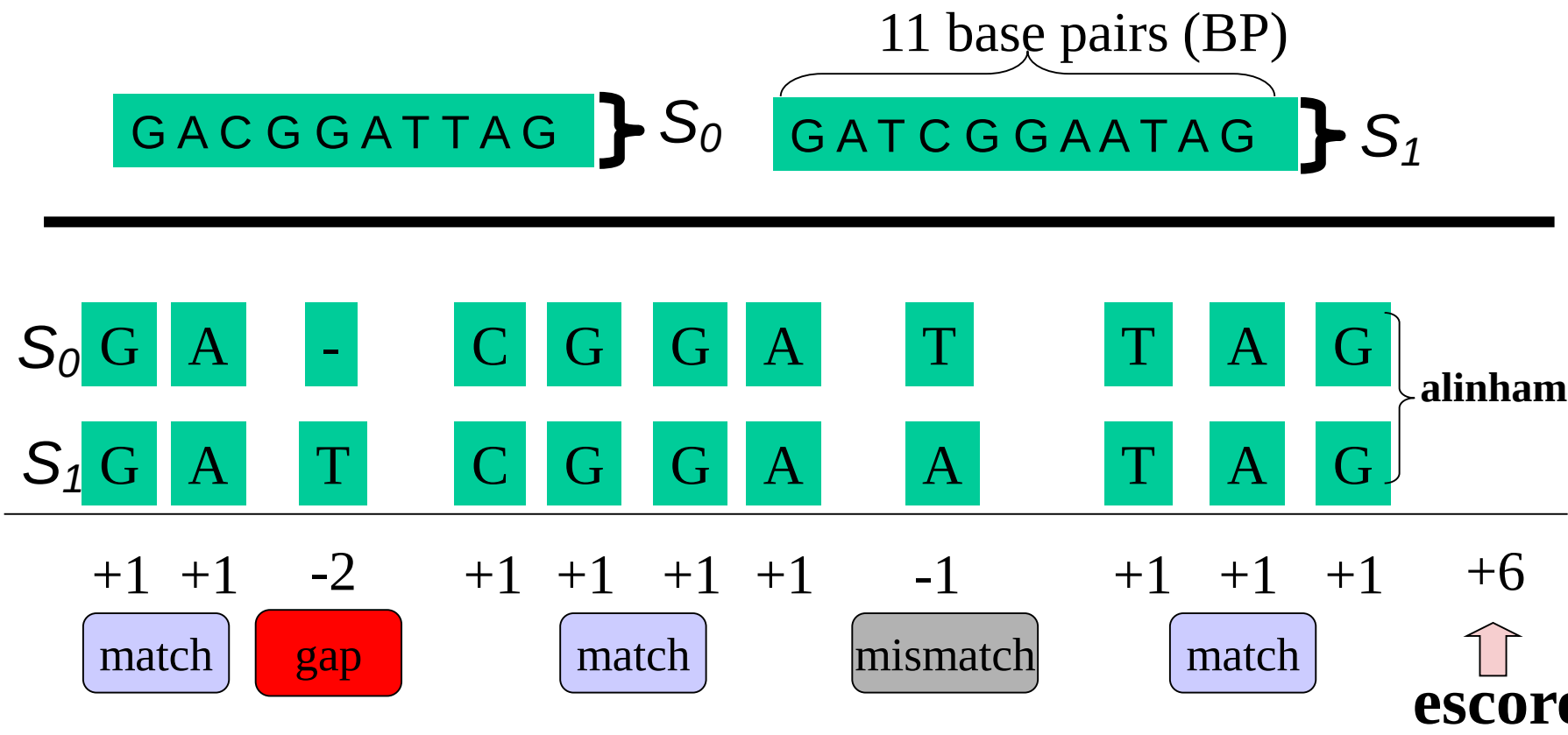
```
res=calcula(1);
/*espera o valor de e */
if (e ==1) /*acertei!*/
   print(res);
else /* errei… */
   res= calcula(e);
   print (res);
```

Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação Pairwise de Sequências Biológicas
- Tarefas independentes: PackageBLAST
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusão

Comparação Pairwise de Sequencias Biológicas

O objetivo da comparação de duas sequências é produzir um escore e um alinhamento



Comparação de Sequências Smith-Waterman

- Smith-Waterman (SW) propuseram um algoritmo baseado em programação dinâmica que obtem o alinhamento local ótimo entre duas sequências de tamanho *n* em tempo e espaço $O(n^2)$.
  - Os tempos de execução podem ser muito grandes e o uso de memória pode ser alto.
    - Para comparar sequencias de DNA de 33 Milhões de caracteres x 46 Milhões, necessitaríamos de 12 PB de memória e mais de 1.000.000.000.000 de operações

Algoritmo Smith-Waterman Alinhamento Local

- Dadas duas sequencias S_0 e S_1, onde $|S_0|=m$ e $|S_1|=n$, a matriz de programação dinâmica $H_{m+1,n+1}$ é calculada com a seguinte equação:

$$H[i,j] = \max \begin{cases} H[i,j-1] - g \\ H[i-1,j] - g \\ H[i-1,j-1] + p(i,j) \\ 0. \end{cases}$$

gap

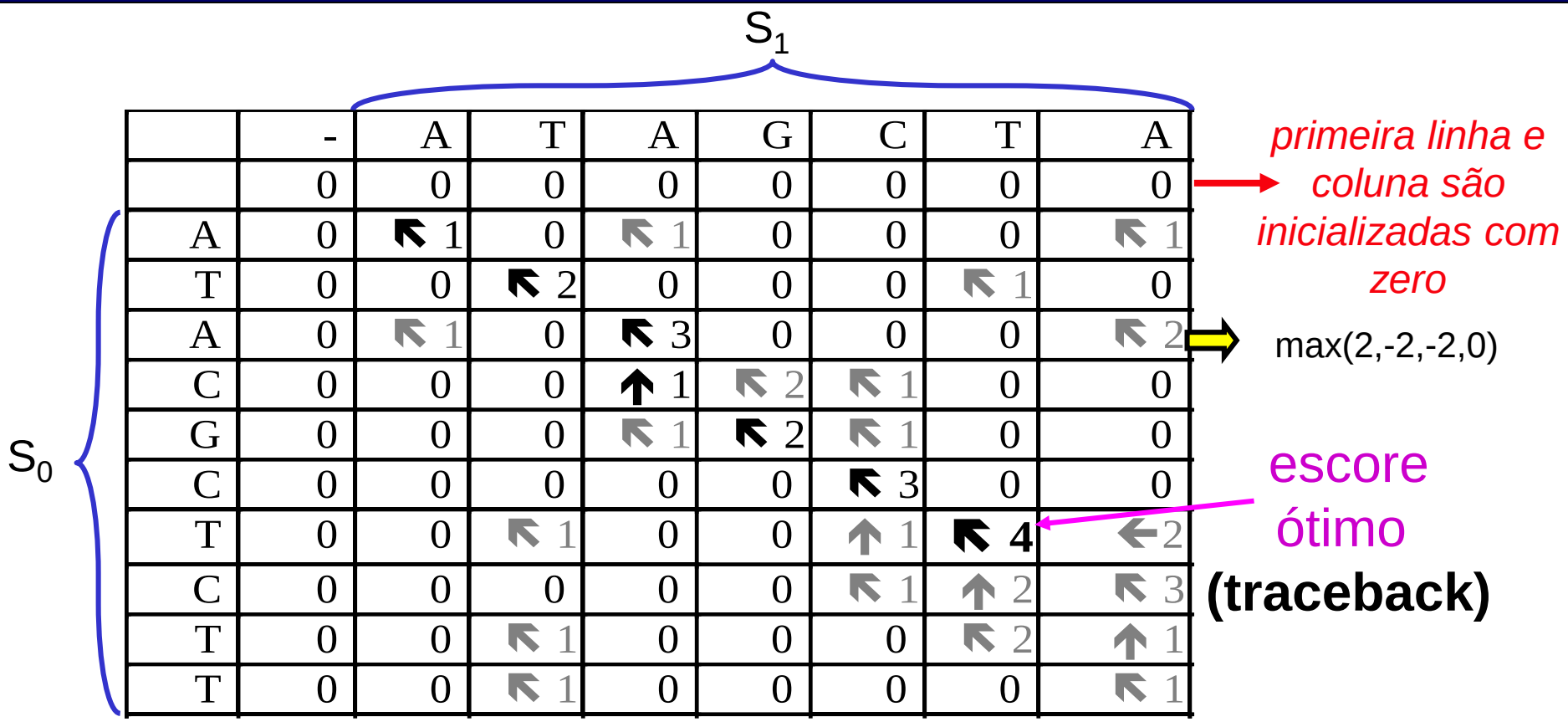
match

$$p(i,j) = \begin{cases} ma, & \text{if } s[i] = t[j] \\ mi, & \text{otherwise} \end{cases}$$

mismatch

Para calcular H[i][j], precisamos ter H[i-1][j], H[i-1][j-1] and H[i][j-1]

Algoritmo Smith-Waterman Exemplo



		-	A	T	A	G	C	T	A
		0	0	0	0	0	0	0	0
A		0	↖ 1	0	↖ 1	0	0	0	↖ 1
T		0	0	↖ 2	0	0	0	↖ 1	0
A		0	↖ 1	0	↖ 3	0	0	0	↖ 2
C		0	0	0	↑ 1	↖ 2	↖ 1	0	0
G		0	0	0	↖ 1	↖ 2	↖ 1	0	0
C		0	0	0	0	0	↖ 3	0	0
T		0	0	↖ 1	0	0	↑ 1	↖ **4**	← 2
C		0	0	0	0	0	↖ 1	↑ 2	↖ 3
T		0	0	↖ 1	0	0	0	↖ 2	↑ 1
T		0	0	↖ 1	0	0	0	0	↖ 1

Local Alignment:

A T A - G C T

A T A C G C T

pontuação: g=-2, mi=-1, ma=1

Comparação de Sequências DIALIGN

- Alinha fragmentos de subsequências.
- Também baseado em programação dinâmica, com complexidade de tempo e memória $O(n^2)$.
- Com as sequências S_0 e S_1, duas matrizes são calculadas: *score e prec.*

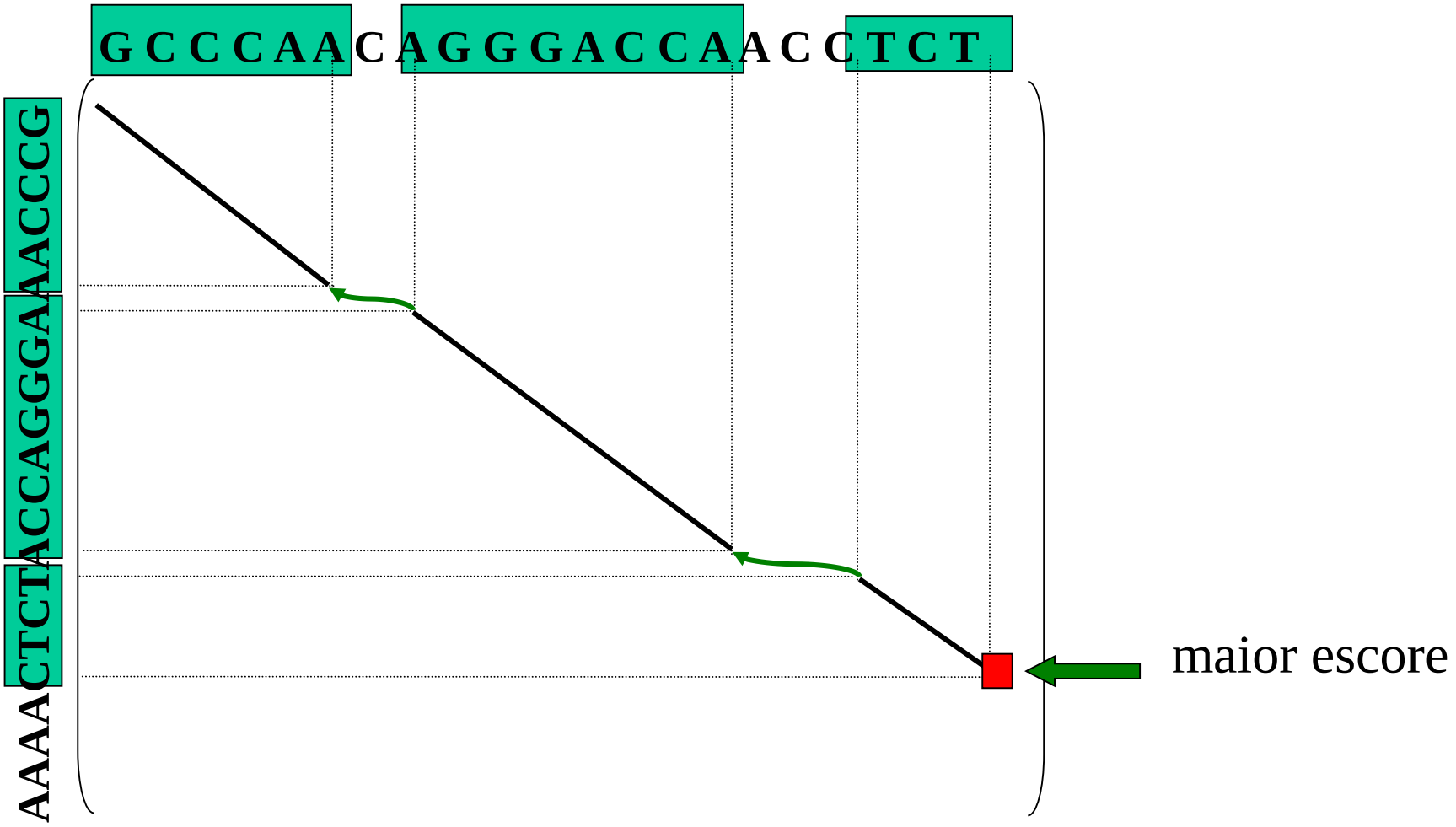
$$score(i,j) = \max \{ score(i-1,j), score(i,j-1), \sigma(D_{i,j}) \}$$

$\sigma(D_{i,j})$ = maior cadeia de fragmentos que terminam em (i,j)

$$prec(i,j) = \begin{cases} prec(i,j-1), \text{ if } score(i,j) = score(i-1,j) \\ prec(i-1,j), \text{ if } score(i,j-1) < score(i,j) = score(i-1,j) \\ D_{i,j}, \text{ if } (score(i,j-1) \text{ and } score(i-1,j)) < score(i,j) = \sigma(D_{i,j}) \end{cases}$$

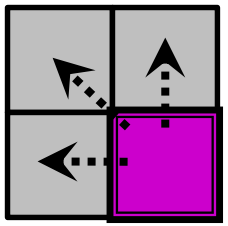
Dependência similar ao Smith-Waterman

Algoritmo DIALIGN



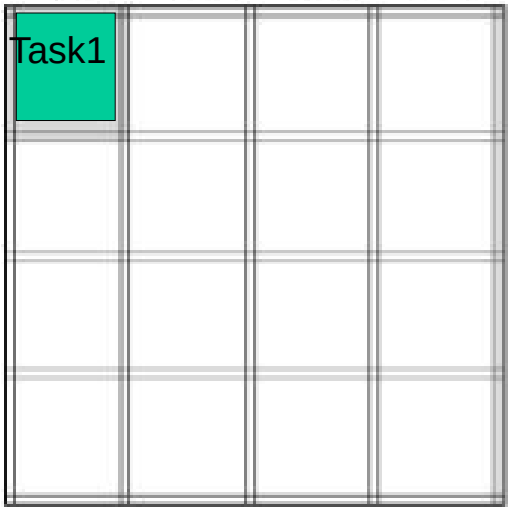

```
Alignment:GCCCAACAGGGACAACCTCT----
          GCCCAA-AGGGACC--CTCTAAAA
```

Algoritmos Smith-Waterman e Dialign – Wavefront Paralelism

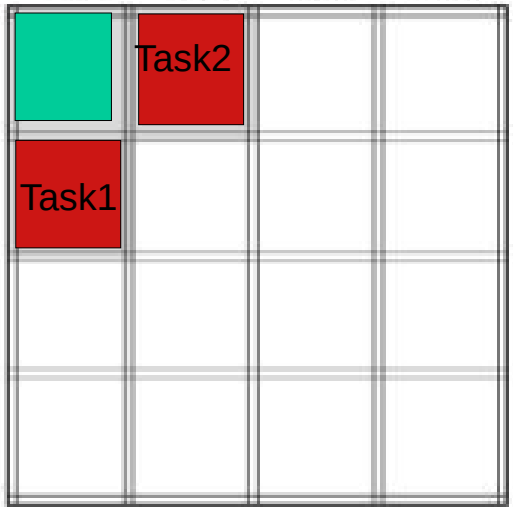


(i,j) depende de (i-1,j); (i-1,j-1) e (i,j-1)

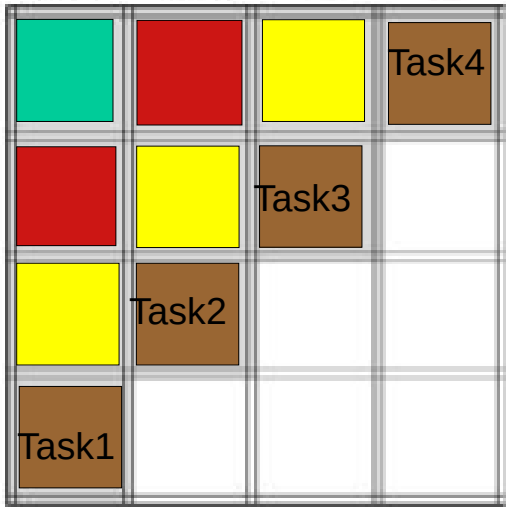
7 (4+4-1) antidiagonais



(a)



(b)

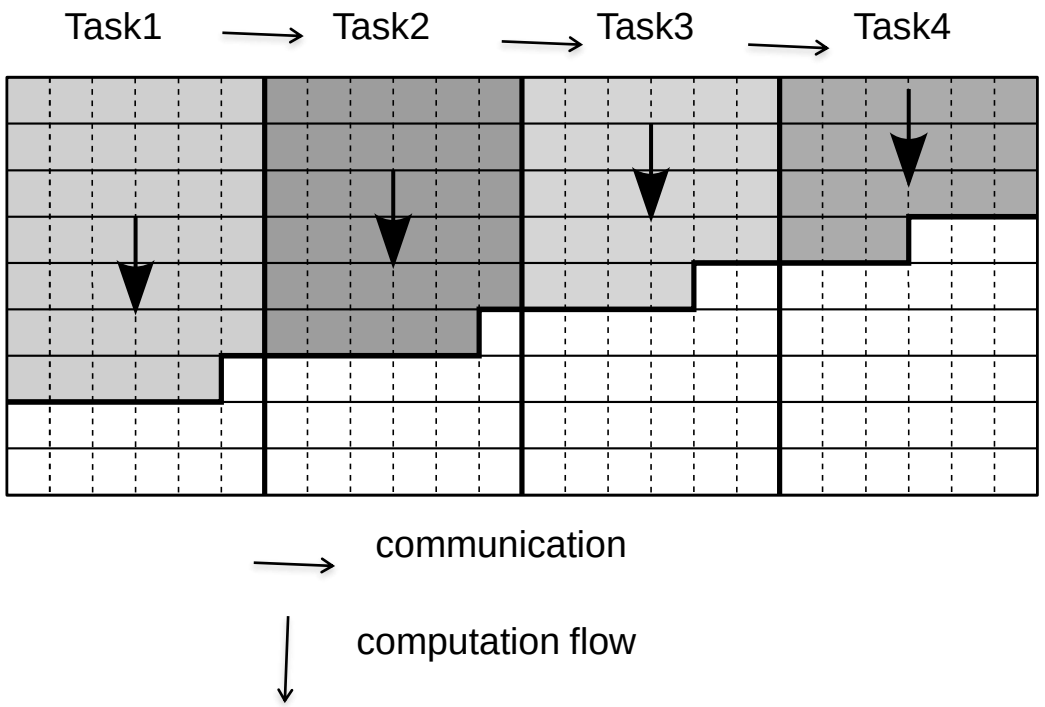


(c)

Paralelismo mínimo

Paralelismo máximo

Algoritmos Smith-Waterman e Dialign – Block Wavefront Paralelism

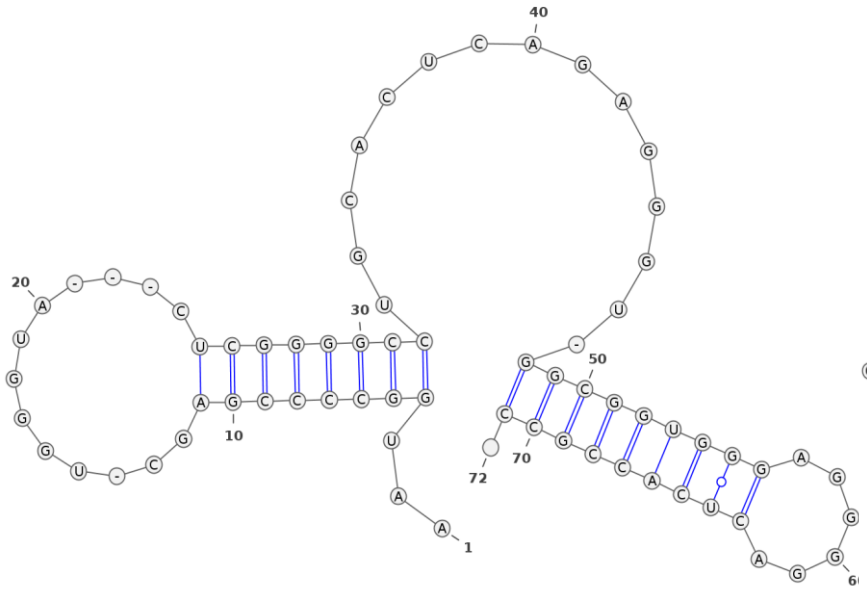


Existem 32 (24+9-1) antidiagonais porém somente 4 tarefas

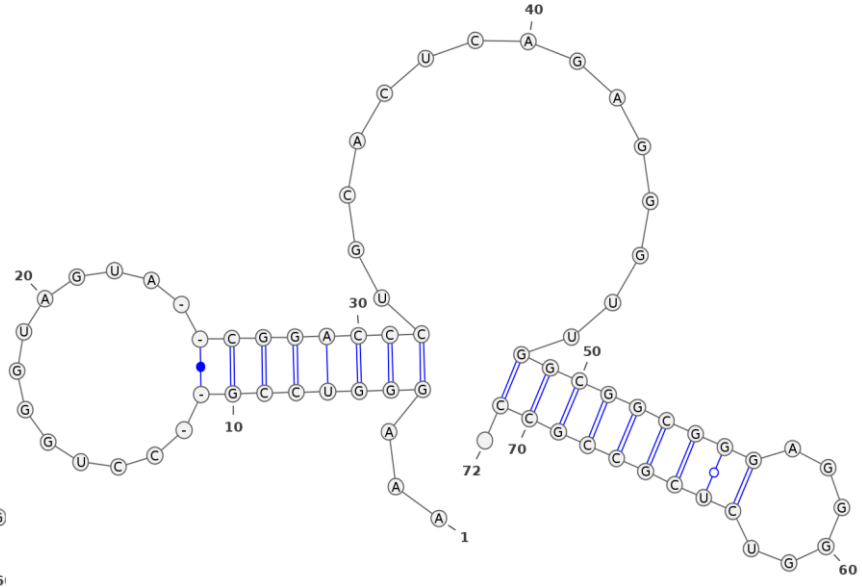
Comparação e Dobramento de Sequências: Sankoff

- Dobra (fold) e alinha duas sequências de RNA simultaneamente em tempo $O(n^6)$ e memória $O(n^4)$

```
AAUGGCCCCGAGC-UGGCUA--CUCGGGGCCUGCACUCAGAGGCU-GGCGCUGGGAGGGGACUCACCGCC   RNA 1
...(((((((((.............)))))))).................(((((((((......)))))))))
AAAGGGUCCG--CCUGGCUAGUA--CGGACCCUGCACUCAGAGGCUUGGCGGCGGGAGGGGUCUCGCCGCC   RNA 2
```



RNA 1



RNA 2

Algoritmo Sankoff

$$S_{i,j,k,l} = \max \begin{cases} S_{i+1,j,k,l} + \gamma & \text{(a)} \\ S_{i,j-1,k,l} + \gamma & \text{(b)} \\ S_{i,j,k+1,l} + \gamma & \text{(c)} \\ S_{i,j,k,l-1} + \gamma & \text{(d)} \\ S_{i+1,j,k+1,l} + \sigma(A_i, B_k) & \text{(e)} \\ S_{i,j-1,k,l-1} + \sigma(A_j, B_l) & \text{(f)} \\ S_{i+1,j-1,k,l} + \beta^A_{ij} + 2.\gamma & \text{(g)} \\ S_{i,j,k+1,l-1} + \beta^B_{kl} + 2.\gamma & \text{(h)} \\ S_{i+1,j-1,k+1,l-1} + \beta^A_{ik} + \beta^B_{kl} + \chi(A_i, A_j, B_k, B_l) & \text{(i)} \\ \max\limits_{\substack{i<m<j \\ k<n<l}} \{S_{i,m,k,n} + S_{m+1,j,n+1,l}\} & \text{(j)} \end{cases}$$

Agenda

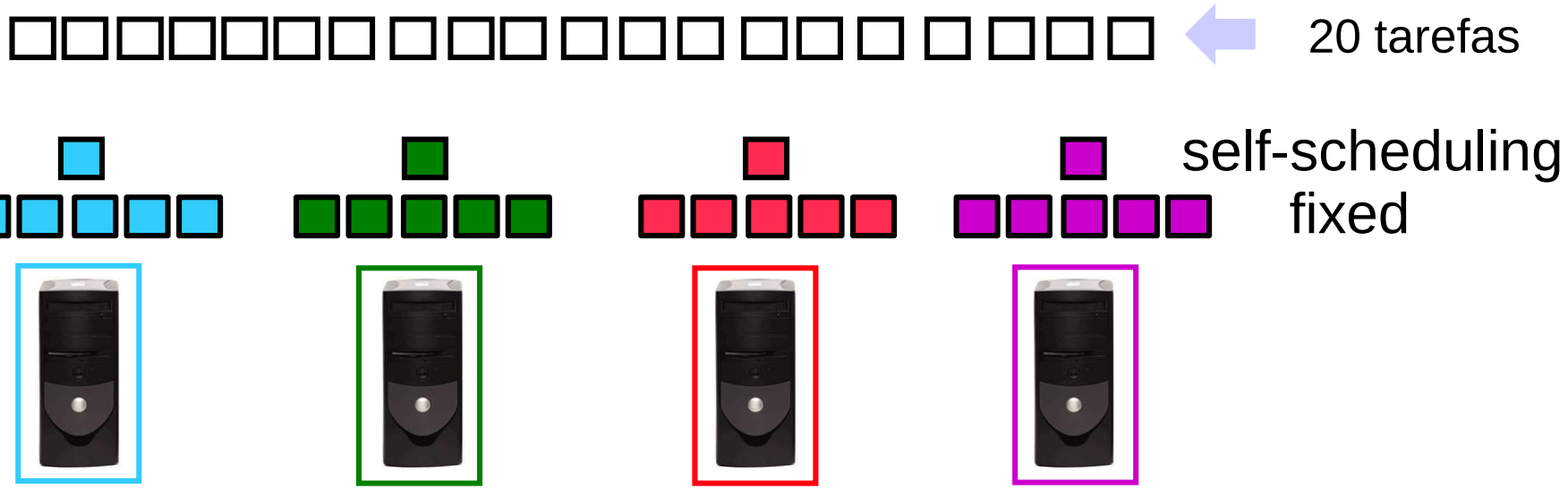
- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação de Sequências Biológicas
- Tarefas independentes: PackageBLAST
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusão

Estudo de Caso em Grid: Tarefas independentes

- *PackageBLAST*: compara uma sequência de proteína com um banco de dados de proteínas (composto de mais de 500.000 sequências) em ambiente de grid heterogêneo (máquinas com configurações diversas) e não-dedicado (a tarefa remota compete pela CPU com as tarefas locais):
  - Múltiplas políticas de alocação
  - Uso de informação histórica

Estudo de Caso em Grid: Tarefas independentes

- Múltiplas políticas de alocação:
  - Fixed: divide o número de tarefas pelo número de máquinas
    - Pouca comunicação, baixo balanceamento de carga
  - Self-Scheduling (SS): atribui 1 tarefa a cada máquina
    - Muita comunicação, alto balanceamento de carga
  - Entre esses extremos: TSS, GSS, FAC2, etc.

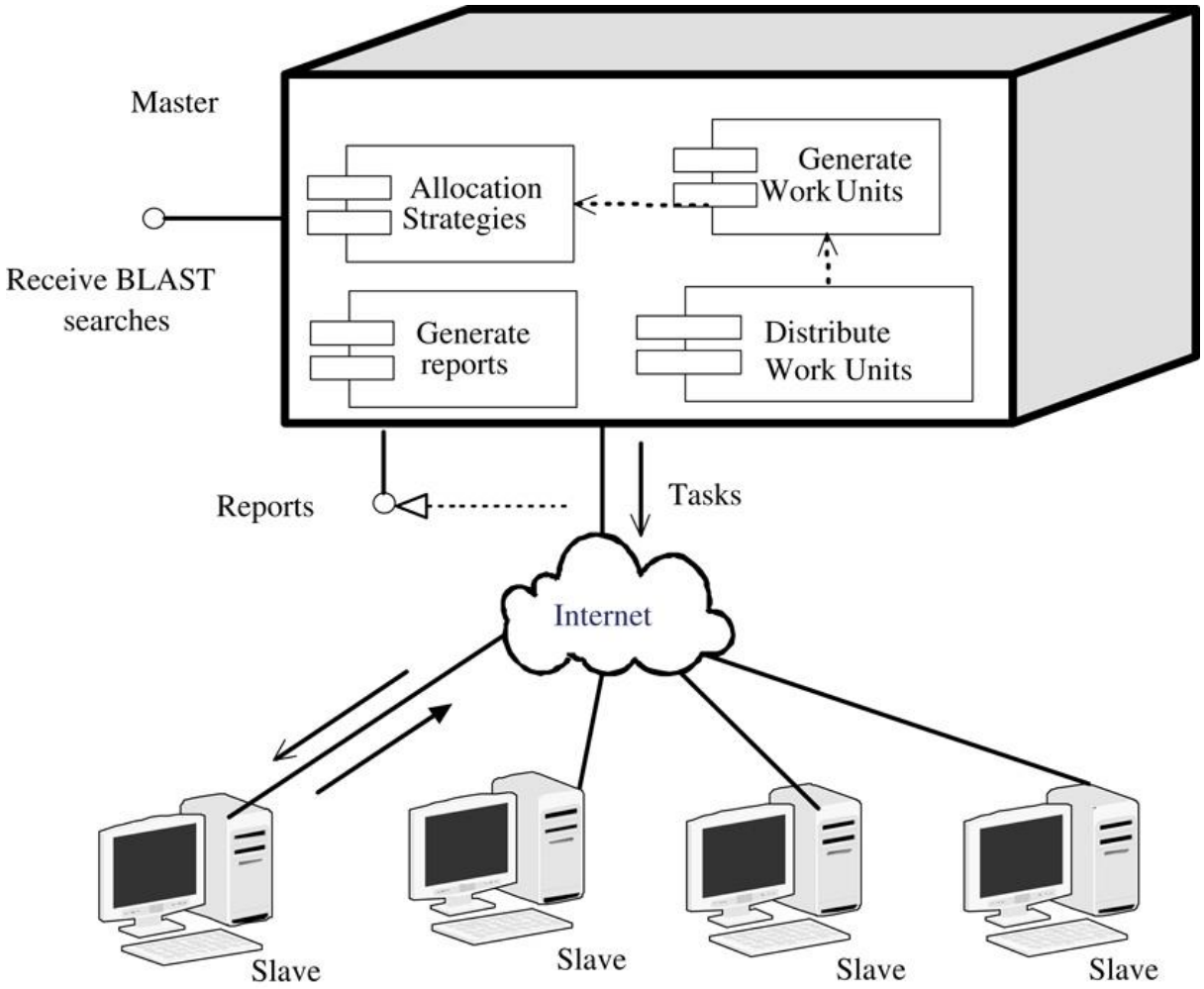


Estudo de Caso em Grid: Tarefas independentes

- Uso de informação histórica:
  - Periodicamente as máquinas enviam informações sobre o avanço da computação
  - Ω: número de notificações que devem ser levadas em conta
    - $\Omega = 3$: as três últimas notificações
  - τ: número de tarefas processadas antes de enviar uma notificação
    - $\tau = 2$: ao processar duas tarefas é enviada uma notificação

Estudo de Caso em Grid: Tarefas independentes

- Arquitetura do PackageBlast



Estudo de Caso em Grid: Speedup

Node names	CPU	RAM memory (MB)	HD (GB)
NB	AMD64 3.2 GHz	512	80
L01, L02, L03	AMD 1.7 GHz	256	30
L04	PII 350 MHz	160	6
P01, P02, P03, P04, P05, P06, P07, P08, P09, P10	AMD 1 GHz	256	20
P11	AMD 900 MHz	128	20
P12	AMD 1.3 GHz	256	40
P13, P14	Pentium 3.0 GHz	512	80

Grid configuration	Machines
2 nodes	NB, L04
4 nodes	NB, L01, P01, L04
8 nodes	NB, L01, L02, L03, P01, P02, P11, L04
16 nodes	NB, L01, L02, L03, P01, P02, P03, P04, P05, P06, P07, P08, P09, P10, P11, L04

J. Parallel Distrib. Comput. 70 (2010) 160–172

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

An adaptive multi-policy grid service for biological sequence comparison

Marcelo S. Sousa [a], Alba C.M.A. Melo [a,b], Azzedine Boukerche [b,*]

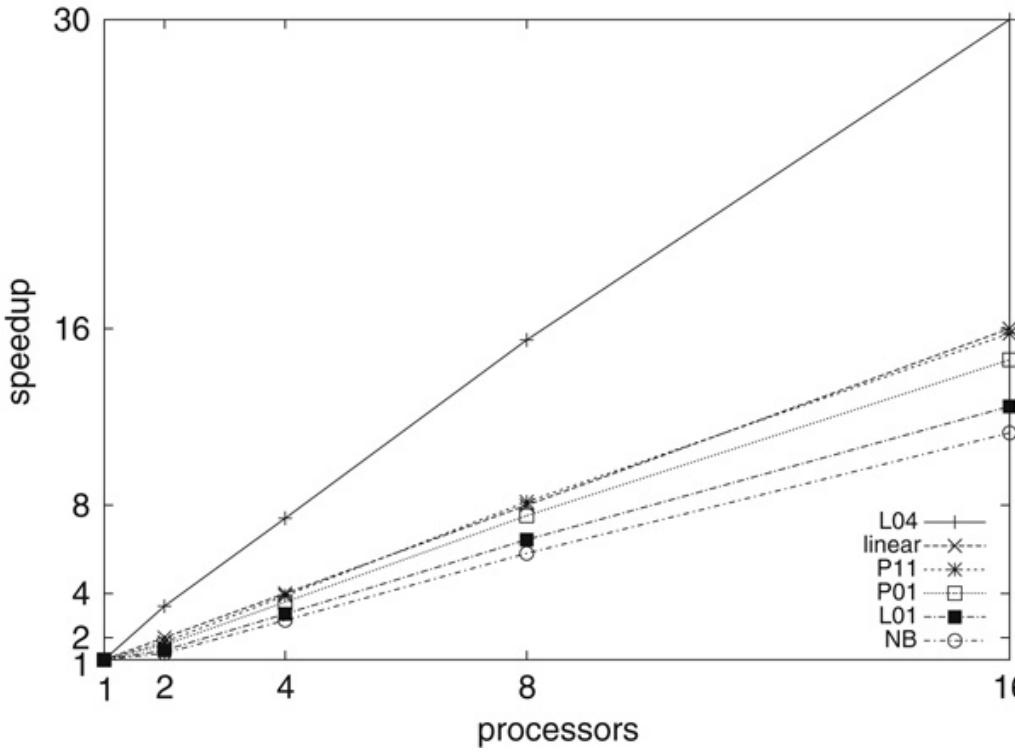
[a] University of Brasilia (UnB), Department of Computer Science, Campus UNB - ICC-Norte - sub-solo, 70910-900 Brasilia-DF, Brazil

[b] University of Ottawa (SITE), PARADISE Research Laboratory, 800 King Edwards, K1N 6N5 Ottawa, Ontario, Canada

PackageBLAST: An Adaptive Multi-Policy Grid Service for Biological Sequence Comparison *

Marcelo S. Sousa
University of Brasilia
Campus UNB - ICC Norte, sub-solo
Brasilia, Brazil
msousa@unb.br

Alba Cristina M. A. Melo
University of Brasilia
Campus UNB - ICC Norte, sub-solo
Brasilia, Brazil
alves@unb.br



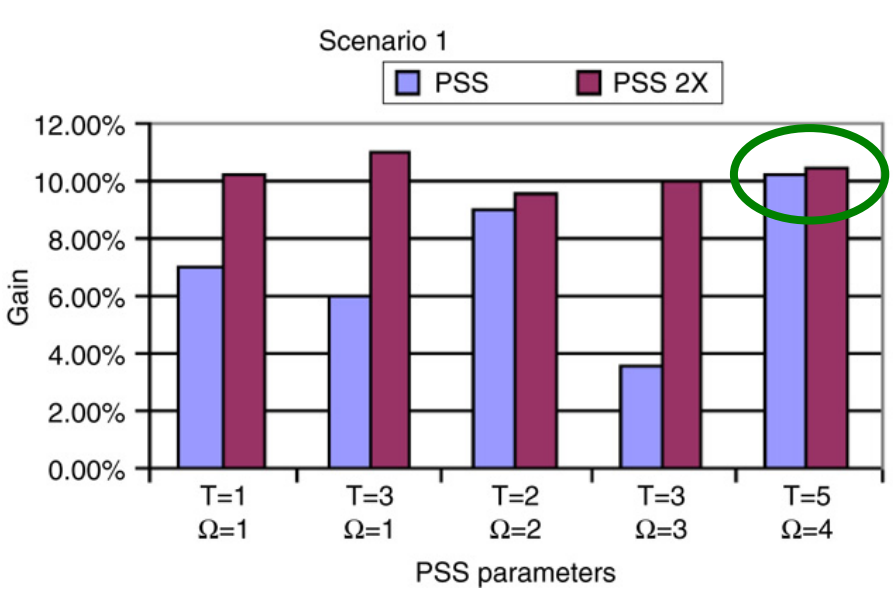
Publicações:

Journal J. Parallel Dist Comp Sys 2010 – A2

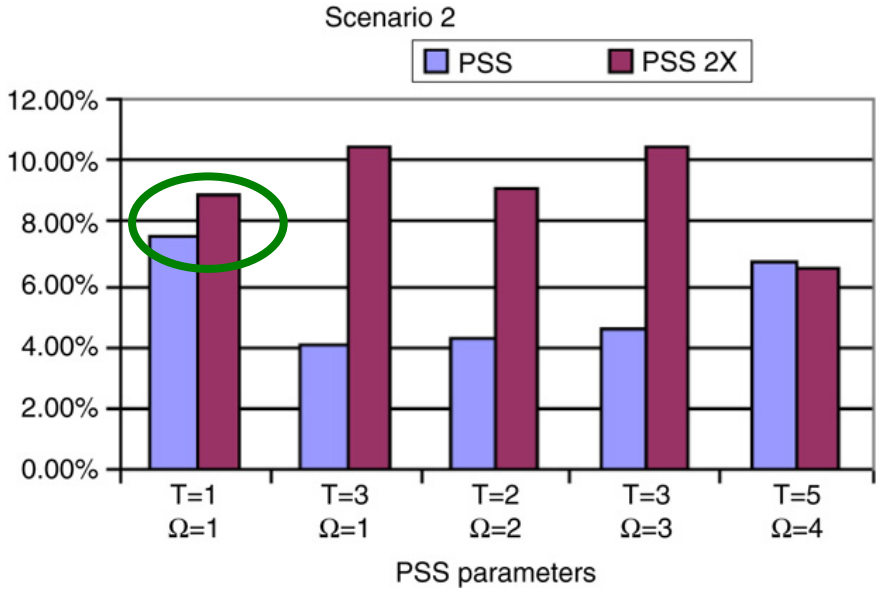
Conferência ACM SAC 2006 – A1

Estudo de Caso em Grid: Impacto das Tarefas Locais

Tarefa local de curta duração



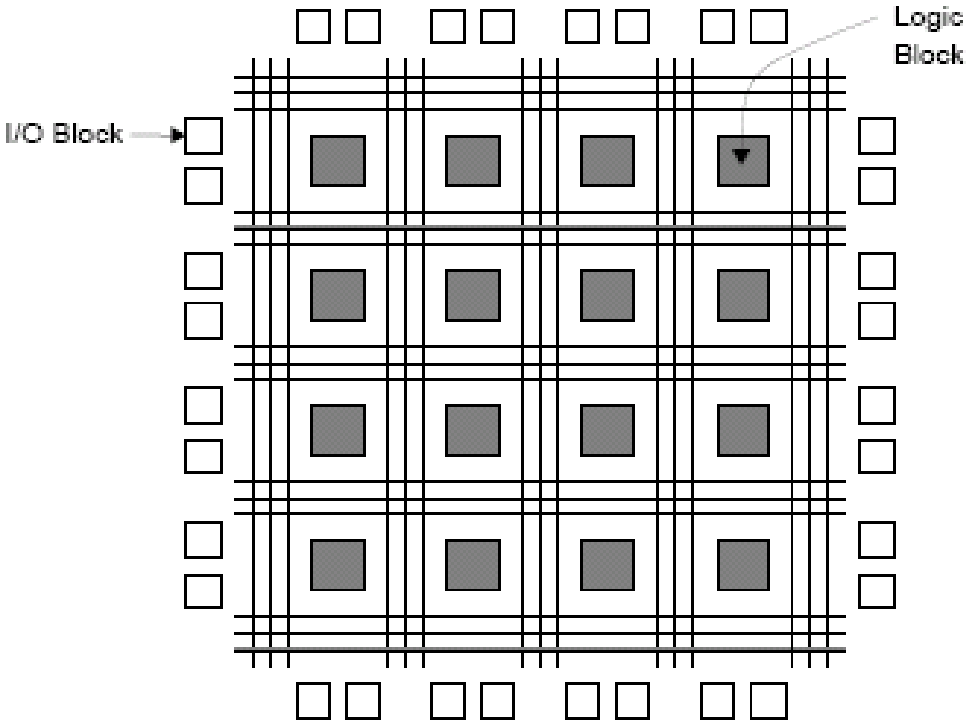
Tarefa local de longa duração



Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação de Sequências Biológicas
- Tarefas independentes: PackageBLAST
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusão

FPGA (Field Programmable Gate Array)



FPGA

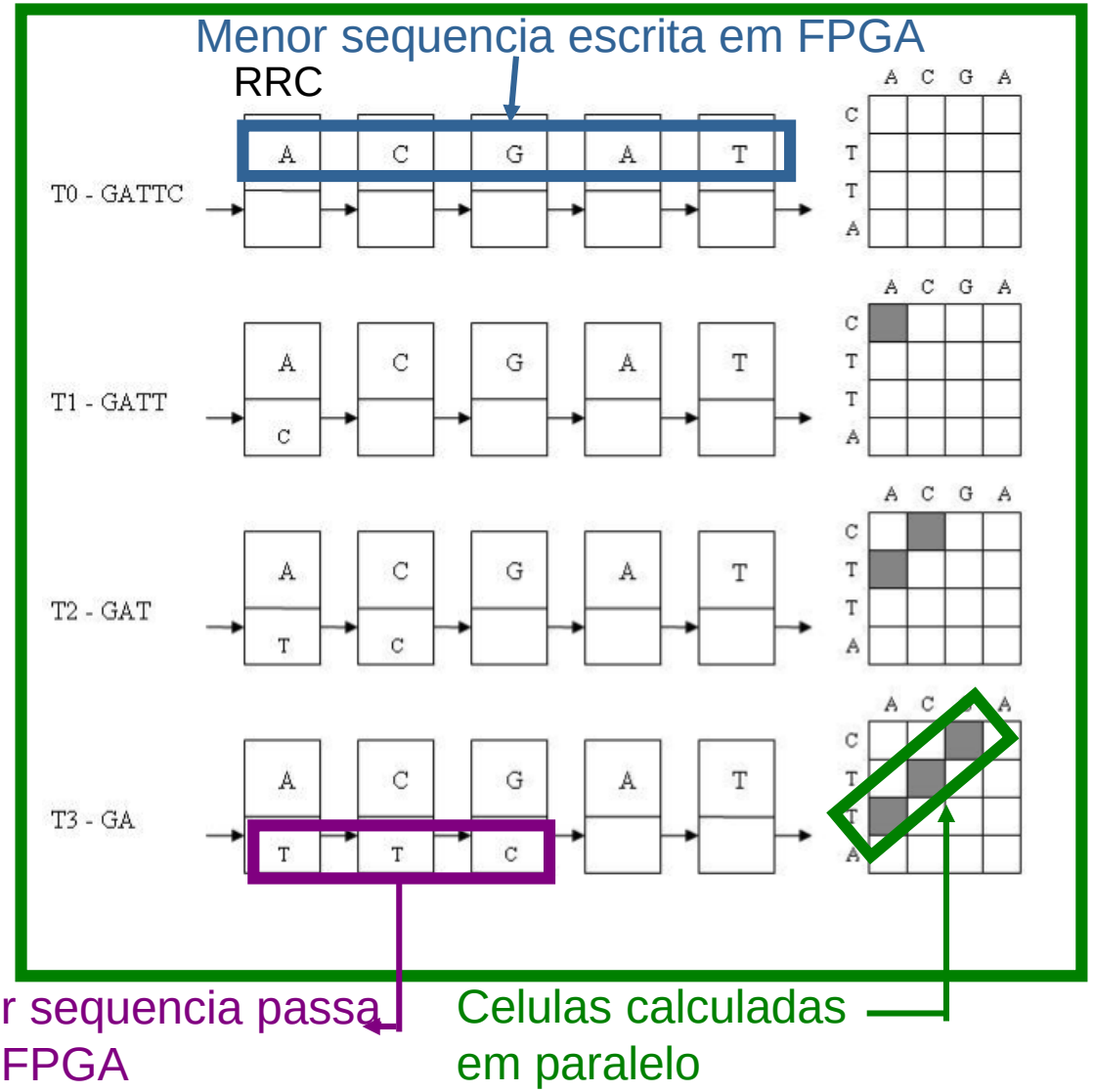
*retirado de S. Brown, J. Rose, “Architecture of FPGAs and CPLDs”, IEEE Design and Test of Computers, 2002

- Hardware programável (VHDL, Verilog, etc)
- Criação de um componente específico para execução de aplicações em hardware.

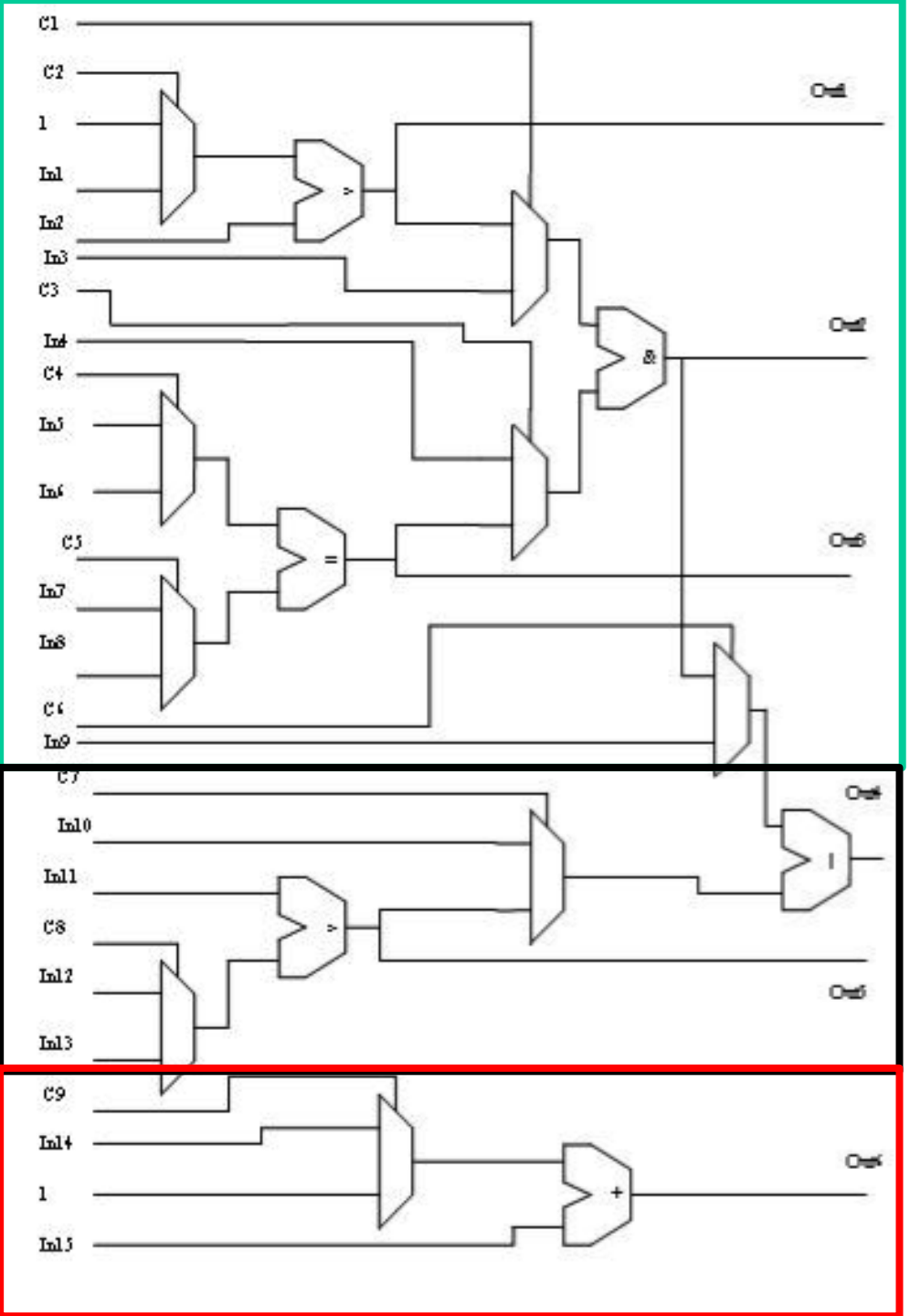
DIALIGN-Align
Fase 1 – Obtem o escore

Foi projetado um vetor de elementos de processamento (PEs), chamados RRC

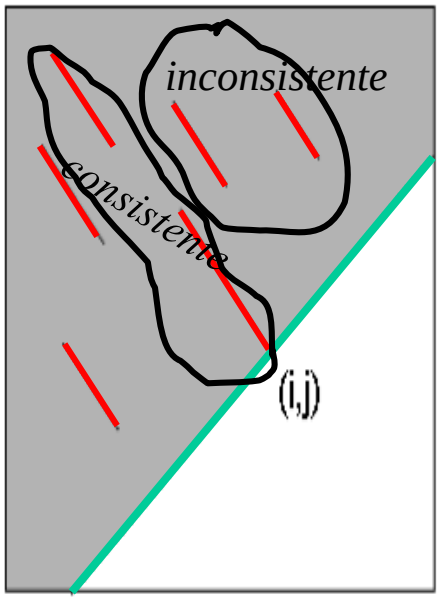
sincronização: cada passo é executado em 1 ciclo de clock



DIALIGN-Align Fase 1



Projeto do RRC



Escores já calculados

Diagonal que está sendo calculada

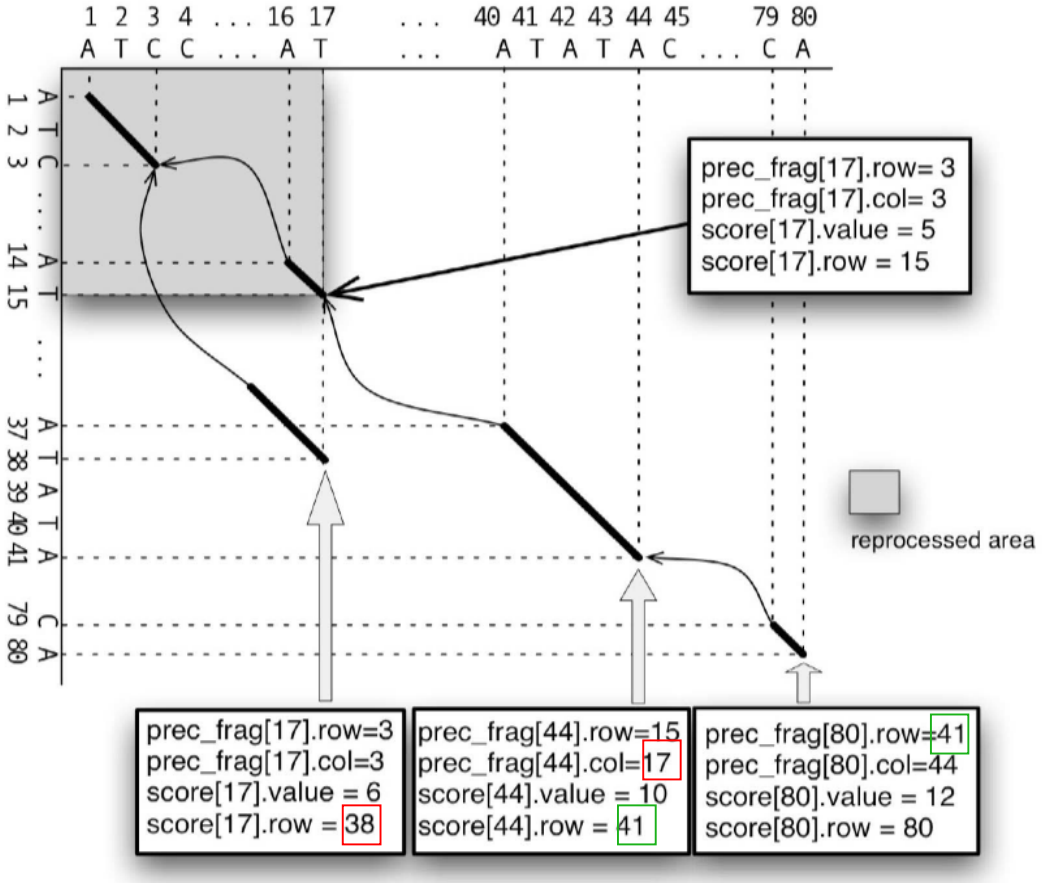
Fragmentos acima do threshold T

Verificação de consistência

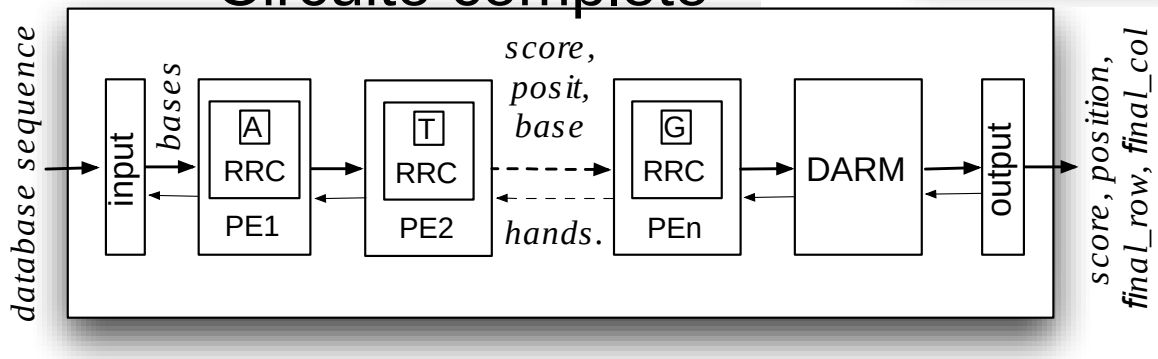
DIALIGN-Align Fase 2 –Alinhamento

Cada RRC calcula 1 coluna e guarda informações sobre o maior escore da sua coluna (especulação)

Na fase 2, os valores especulados são usados. Em caso de erro, reprocessa



Circuito completo



DIALIGN-Align Resultados

Id. RNA	Id. DNA	T	Rounds	Time FPGA	Time CPU	Speedup
MI0000061	AM270408	0	12	0.020441s	1.57s	76.80
MI0000061	AM270375	0	12	0.028396s	2.23s	78.53
MI0000061	AL590443	0	12	0.032188s	2.59s	80.46
MIR156d|stem	AM236080	0	11	0.809720s	102.52s	126.61
MIR156d|stem	AL590443	0	15	0.001025s	4.05s	130.40
SS128	SS10000000	0	10	1.599409s	226.14s	141.38

Publicação:
IEEE Transactions Computers 2010 – A1

A Hardware Accelerator for the Fast Retrieval of DIALIGN Biological Sequence Alignments in Linear Space

Azzedine Boukerche, *Senior Member*, *IEEE*, Jan M. Correa,
Alba Cristina M.A. de Melo, *Senior Member*, *IEEE*, and Ricardo P. Jacobi

Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação de Sequências Biológicas
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusões e Trabalhos Futuros

Progamação em GPU

- Em CUDA, temos um programa que roda em CPU e as rotinas invocadas em GPU são chamadas kernel.
- O mesmo kernel pode ser executado por vários blocos (B) que contém cada um várias threads (T).
- Todas as threads de um mesmo bloco compartilham a *shared memory* (rápida).
- Todas as threads compartilham a memória global (lenta).
- Os blocos são executados em paralelo (MIMD) e as threads de um mesmo bloco são executadas no modelo SIMT (*Single Instruction Multiple Thread*).
- Sincronização:
  - ao final da invocação do kernel
  - Threads do mesmo bloco: __*syncthreads*

Arquitetura GPU NVidia Pascal

60 SMs cada um com 64 cores = 3840 cores

* Retirado de NVidia Tesla P100 white paper

CUDAlign: Objetivo e Evolução

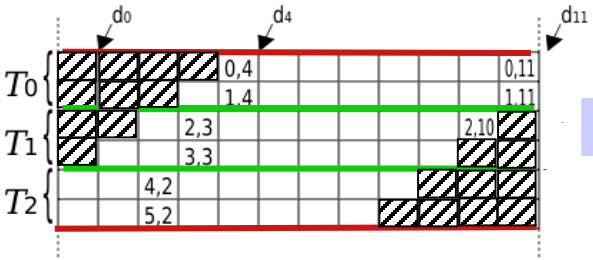
- Objetivo: comparar duas sequências de DNA muito longas em GPU com Smith-Waterman
  - CUDAlign 1.0: escore em uma GPU
  - CUDAlign 2.0: escore e alinhamento em uma GPU
  - CUDAlign 2.1: escore e alinhamento em uma GPU com pruning
  - CUDAlign 3.0: escore em várias GPUs
  - CUDAlign 4.0: escore e alinhamento em várias GPUs
- Tese de Edans F. O. Sandes (Prêmio Capes de Tese na Área de Ciência da Computação 2016)

CUDAlign 1.0: Obtem o escore em 1 GPU

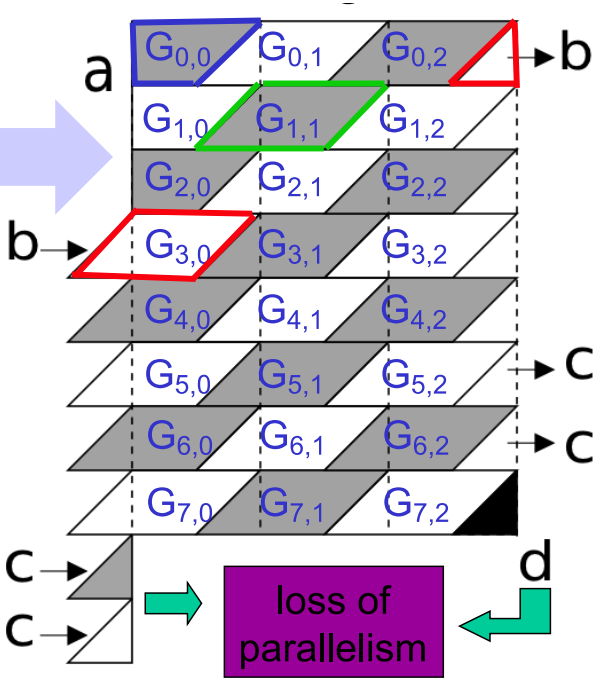
Retangulos:
Abordagem tradicional

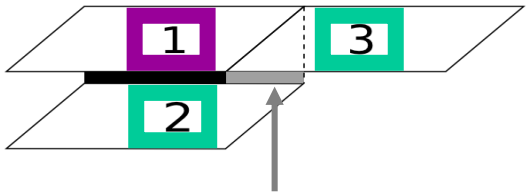
$G_{0,0}$	$G_{0,1}$	$G_{0,2}$
$G_{1,0}$	$G_{1,1}$	$G_{1,2}$
$G_{2,0}$	$G_{2,1}$	$G_{2,2}$
$G_{3,0}$	$G_{3,1}$	$G_{3,2}$
$G_{4,0}$	$G_{4,1}$	$G_{4,2}$
$G_{5,0}$	$G_{5,1}$	$G_{5,2}$
$G_{6,0}$	$G_{6,1}$	$G_{6,2}$
$G_{7,0}$	$G_{7,1}$	$G_{7,2}$
$G_{8,0}$	$G_{8,1}$	$G_{8,2}$

D_5

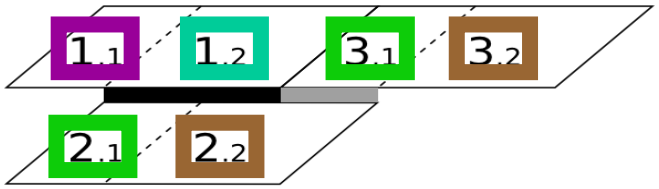


Paralelogramo





Condição de corrida



Solução

CUDAlign 1.0: Invocação dos kernels em GPU

```
function ProcessGrid (B,T,α)
  grid = (B,1,1);
  threads = (T,1,1);
  D = B + (m/αT)-1;
  for Dk = 0 to |D|+B-1 do
    kernel.1 <<<grid,threads>>> (Dk);
    kernel.2 <<<grid,threads>>> (Dk);
  end for
  max, max_pos = GetMaxScore();
  return (max, max_pos);
end function
```

short phase in GPU

long phase in GPU

CUDAlign 1.0: Resultados

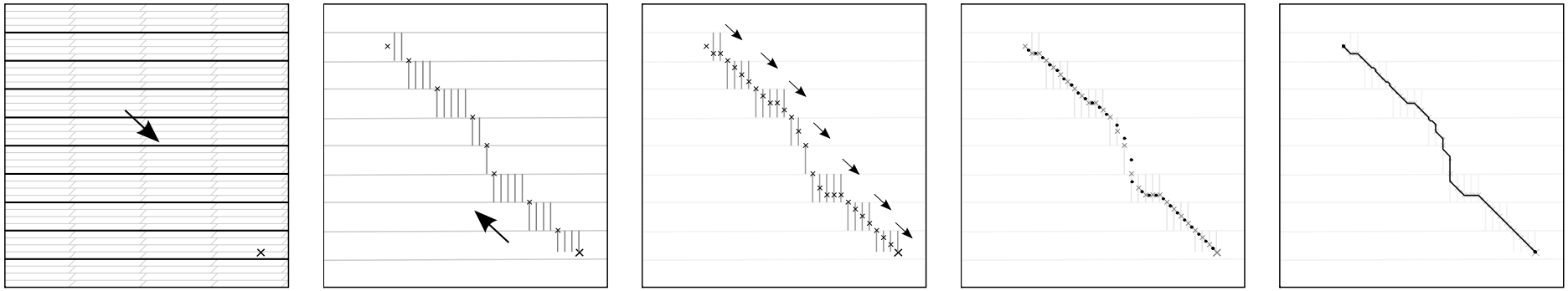
Comparison	8600 GT		GTX 280	
	Time	MCUPS	Time	MCUPS
128K × 128K	8.6s	1895	1.1s	15277
256K × 256K	34.1s	1923	3.7s	17837
512K × 512K	2min15s	1939	13.7s	19147
1000K × 1000K	8min34s	1947	50.6s	19773
2000K × 2000K	34min10s	1951	3min19s	20106
3147K × 3283K	1h28min11s	1953	8min32s	20196
5227K × 5229K	3h53min2s	1955	22min28s	20278
7146K × 5227K	5h18min40s	1954	30min41s	20289
23012K × 24544K	-	-	7h42min10s	20367
32799K × 46944K	-	-	20h59min31s	20375

CUDAlign: Using GPU to Accelerate the Comparison of Megabase Genomic Sequences

Publicação:
Conferência ACM PPoPP 2010 – A1

Edans Flavius de O. Sandes Alba Cristina M. A. de Melo
University of Brasilia (UnB), Brazil
{edans,albamm}@cic.unb.br

CUDAlign 2.0: Escore e alinhamento em 1 GPU

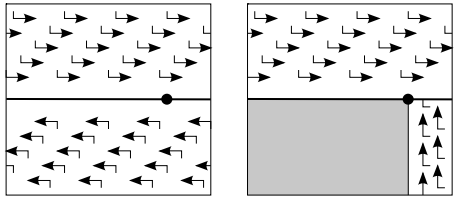


Estágio1 - GPU | Estágio2 - GPU | Estágio3 - GPU | Estágio4 - CPU | Estágio5 - CPU

escore

alinhamento

Comparison	Stages					Total
	1	2	3	4	5+6	
162K x 172K	1.5	<0.1	<0.1	<0.1	<0.1	1.8
543K x 536K	13.6	<0.1	<0.1	<0.1	<0.1	13.9
1044K x 1073K	51.6	3.1	1.0	5.4	0.1	61.6
3147K x 3283K	448	0.1	<0.1	0.3	<0.1	449
5227K x 5229K	1185	65.9	20.3	47.6	1.9	1321
7146K x 5227K	1604	<0.1	<0.1	<0.1	<0.1	1605
23012K x 24544K	23750	0.3	<0.1	0.7	<0.1	23755
32799K x 46944K	65153	805	236	376	9	66579



Execução ortogonal

2011 IEEE International Parallel & Distributed Processing Symposium

CUDAlign 2.0: Obtenção do alinhamento
32MBP x 47 MBP em 18.5 horas

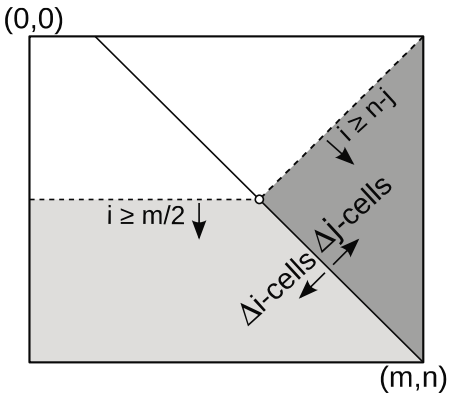
Publicação:
Conferência IEEE IPDPS 2011 – A1

Smith-Waterman Alignment of Huge Sequences with GPU in Linear Space

Edans Flavius de O. Sandes, Alba Cristina M. A. de Melo
Department of Computer Science
University of Brasilia (UnB)
Brasilia, Brazil
{edans,albamm}@cic.unb.br

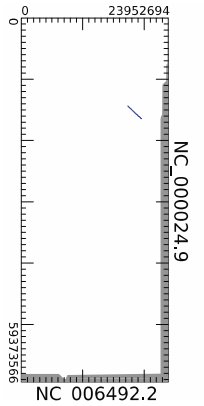
CUDAlign 2.1: Escore e alinhamento em 1 GPU com pruning

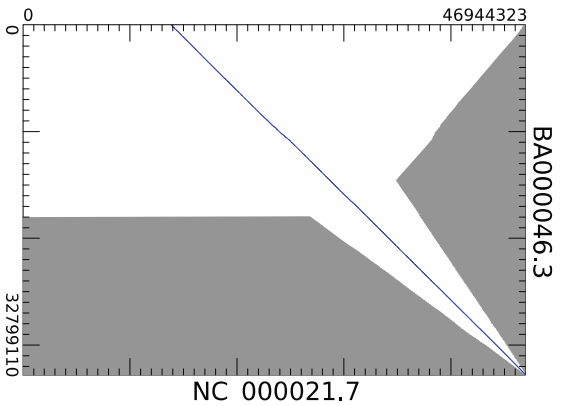
Block pruning



Comparison	Original			Block Pruning				
	Time(s)	Cells	GCUPS	Time(s)	Cells	$GCUPS_P$	GCUPS	Gain
162K×172K	1.2	2.79E+10	23.28	1.2	2.78E+10	23.37	23.45	0.7%
543K×536K	10.8	2.91E+11	26.97	10.8	2.91E+11	26.98	27.00	0.1%
1044K×1073K	40.3	1.12E+12	27.81	36.2	9.97E+11	27.54	30.95	10.1%
3147K×3283K	363.6	1.03E+13	28.41	363.2	1.03E+13	28.40	28.44	0.1%
5227K×5229K	962.4	2.73E+13	28.40	469.5	1.27E+13	26.95	58.21	51.2%
7146K×5227K	1309	3.74E+13	28.53	1309	3.73E+13	28.53	28.54	<0.1%
23012K×24544K	19701	5.65E+14	28.67	19694	5.65E+14	28.67	28.68	<0.1%
59374K×23953K	49634	1.42E+15	28.65	46869	1.34E+15	28.51	30.34	5.6%
32799K×46944K	53869	1.54E+15	28.58	29133	7.99E+14	27.44	52.85	45.9%

CUDAlign 2.1: Obtenção do alinhamento
32MBP x 47 MBP em 8.09 horas
(14.96 horas sem block pruning)





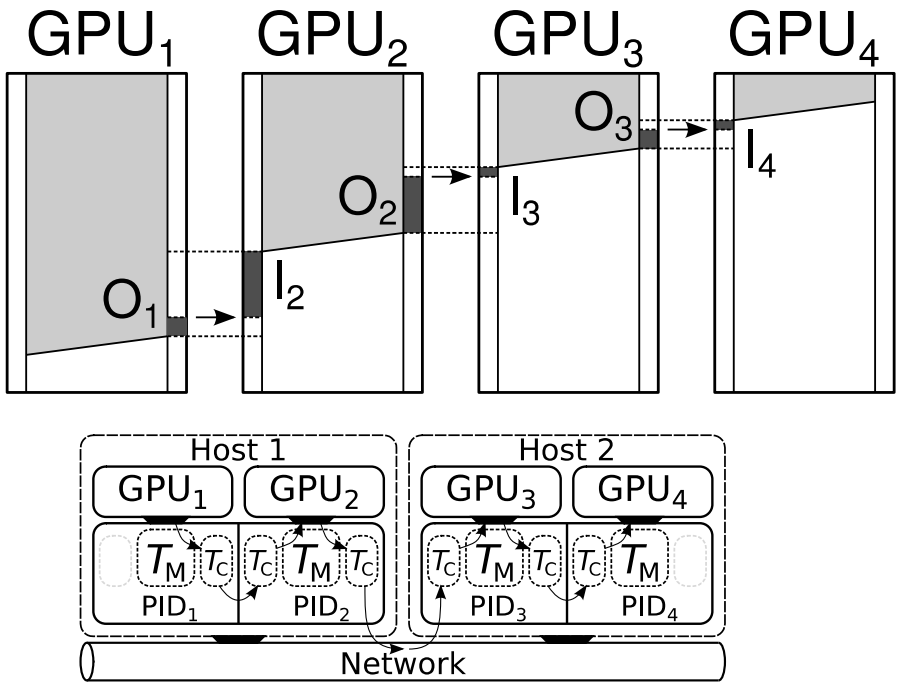
IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 24, NO. 5, MAY 2013 1009

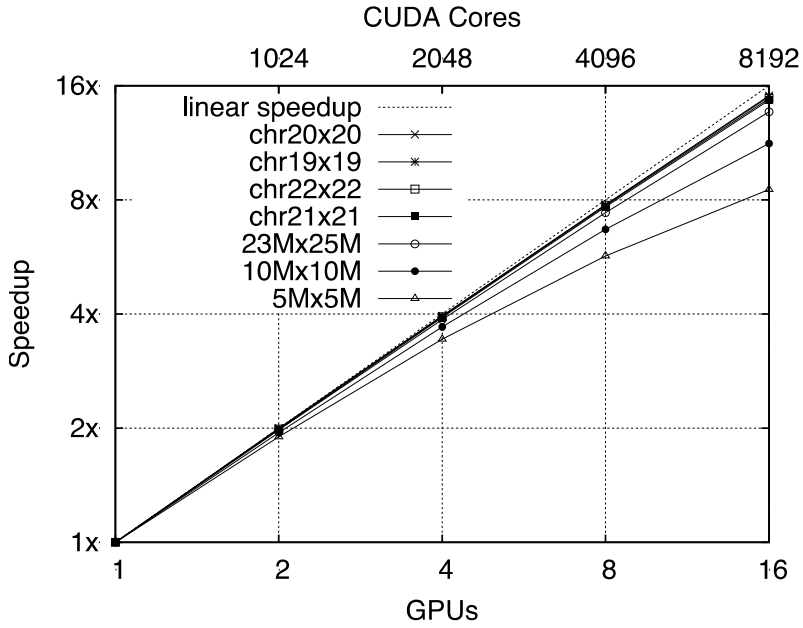
Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU

Edans Flavius de O. Sandes and Alba Cristina M.A. de Melo, *Senior Member, IEEE*

Publicação:
IEEE Transactions Par Dist Syst 2013 – A1

CUDAlign 3.0: Escore em várias GPUs





CUDAlign 3.0: Obtenção do Escore 32MBPx47MBP com 16 GPUs em 1.18 hora

2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

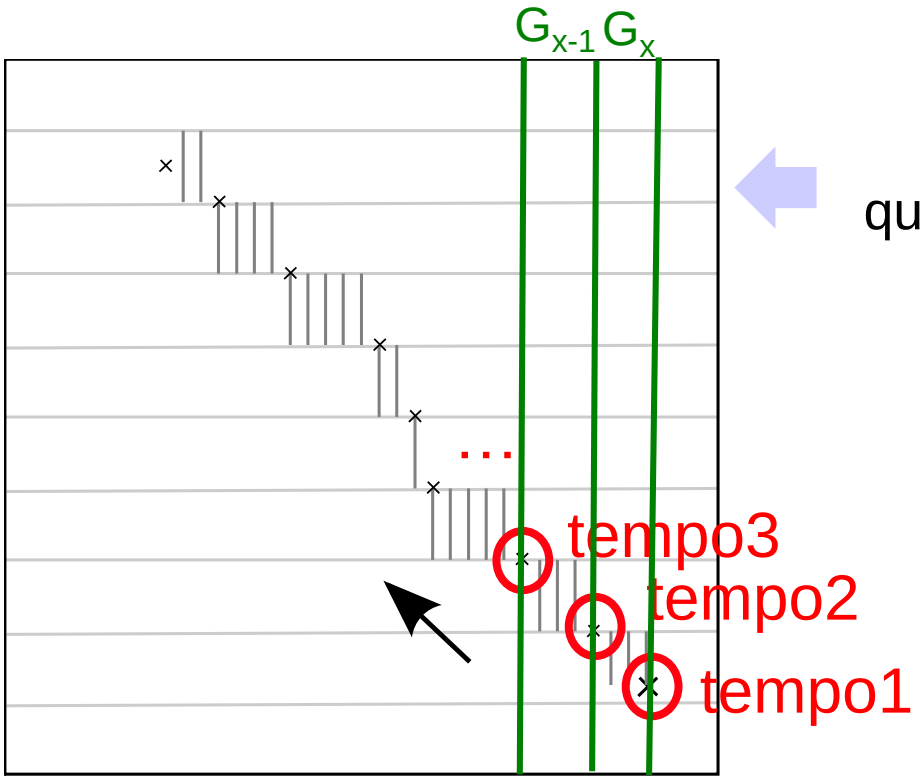
CUDAlign 3.0: Parallel Biological Sequence Comparison in Large GPU Clusters

Edans F. de O. Sandes[□], Guillermo Miranda[†], Alba C. M. A. de Melo[□], Xavier Martorell[†‡], Eduard Ayguadé[†‡]
[□]*University of Brasilia (UnB)*
{edans, albamm}@cic.unb.br
[†]*Barcelona Supercomputing Center (BSC)*
{guillermo.miranda, xavier.martorell, eduard.ayguade}@bsc.es
[‡]*Universitat Politècnica de Catalunya (UPC)*
{xavim, eduard}@ac.upc.edu

Publicação:
Conferência IEEE/ACM CCGRID 2014 – A1

CUDAlign 4.0: Escore e alinhamento em várias GPUs

- Desafio:
  - Paralelização de código inerentemente sequencial (Estágio 2 do CUDAlign)

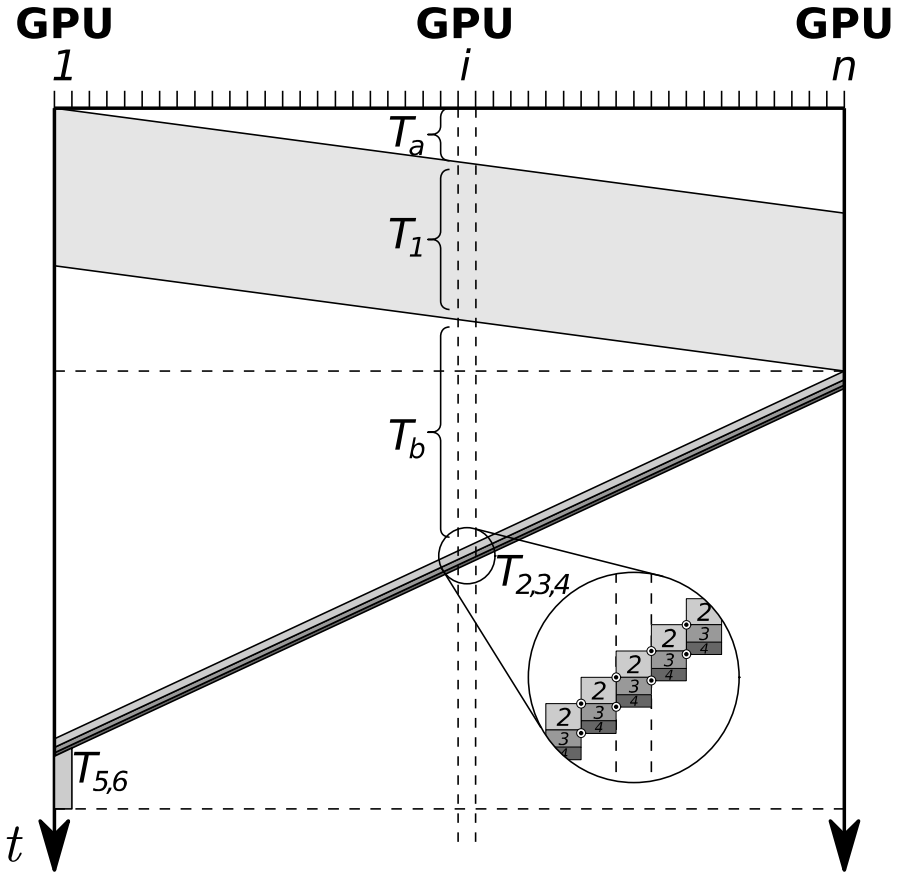


GPU_{x-1} só começa a execução quando a GPU_x calcular o crosspoint

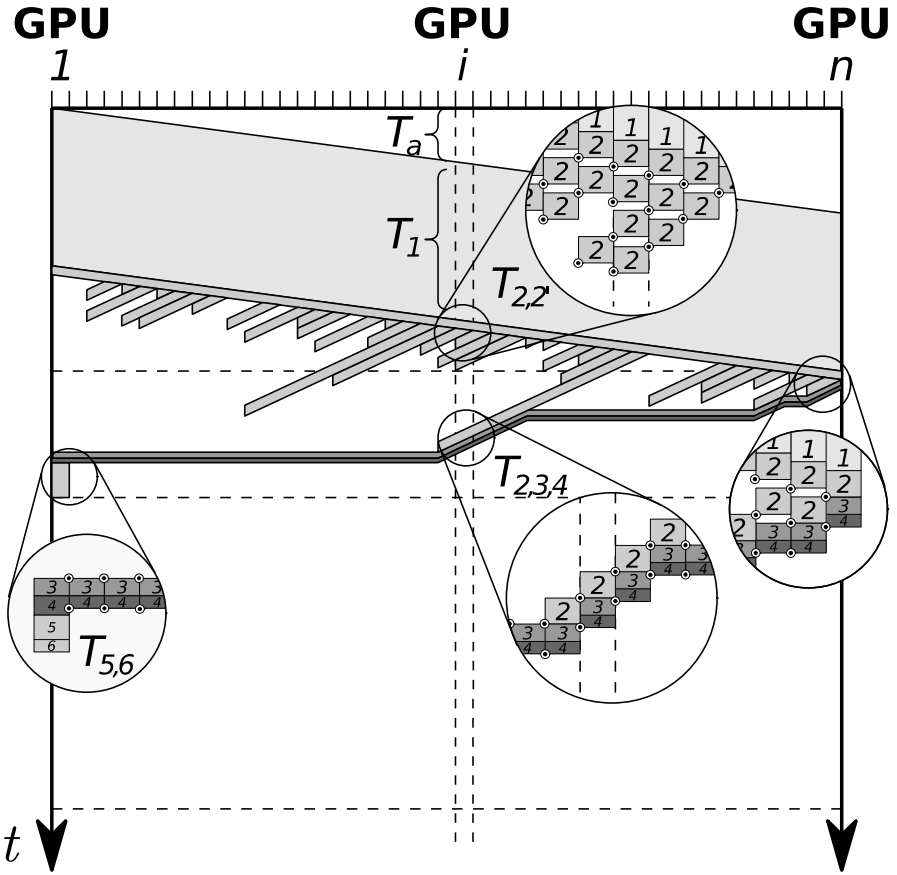
Solução:
Especulação, onde o valor especulado é o valor mais alto calculado pela GPU_{x-1} no estágio 1

CUDAlign 4.0: Incremental Speculative Traceback (IST)

Sem especulação

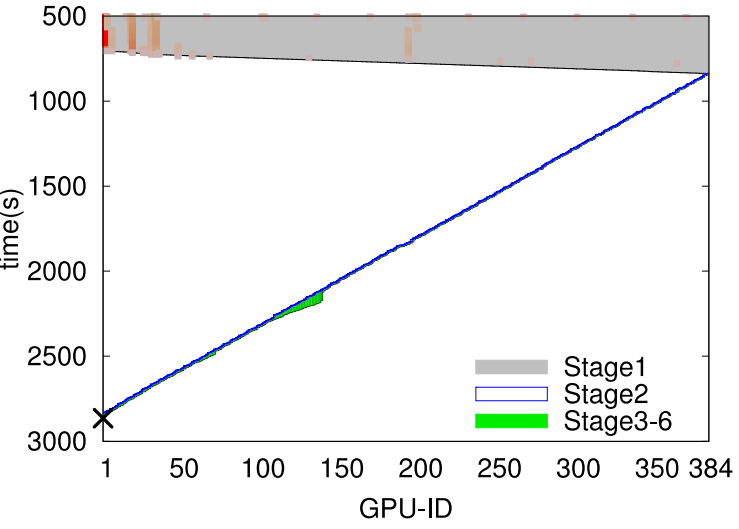


Com especulação

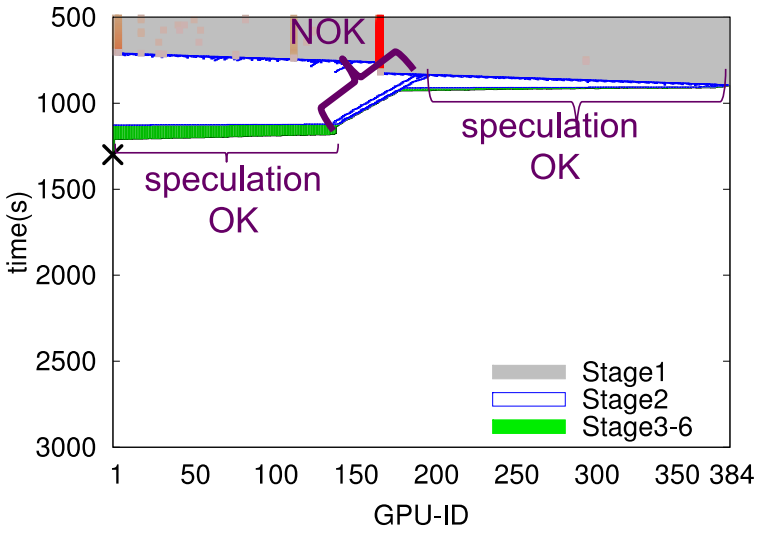


CUDAlign 4.0: Escore e alinhamento em várias GPUs

Without IST



With IST



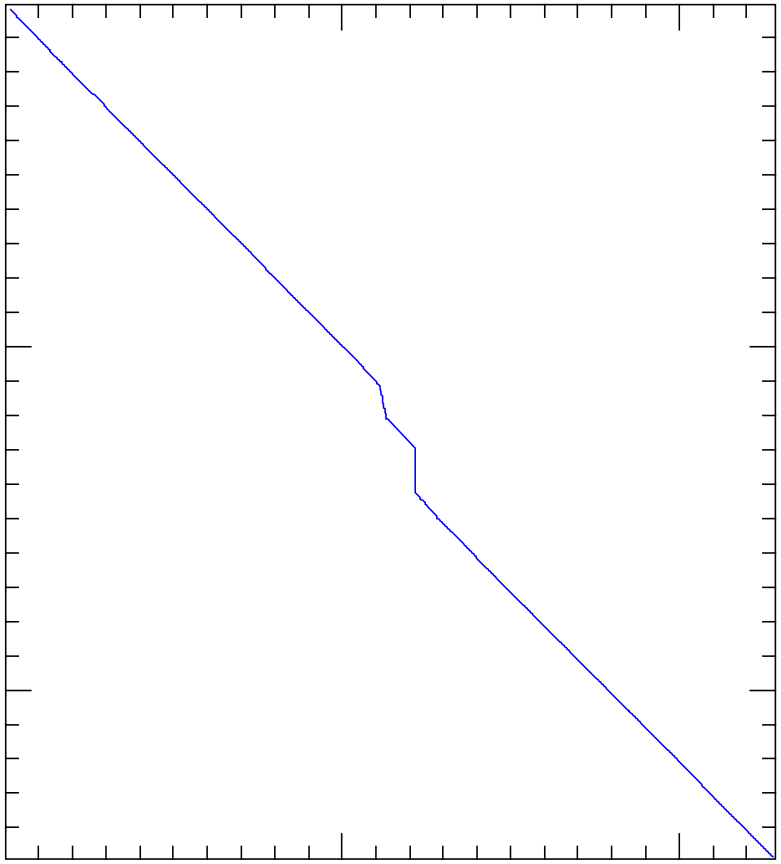
	Cmp.		Total Time (TCUPS)	Stage 1 Time (TCUPS)	Traceback Time (PT/IST)	
128 nodes (384 GPUs)	chr22	PT	993s (2.57)	307s(8.31)	686s	2.61x
		IST	569s (4.49)	306s(8.33)	263s	
	chr16	PT	2870s (2.83)	838s(9.70)	2033s	4.94x
		IST	1305s (6.23)	894s(9.10)	412s	
	chr13	PT	4176s (3.18)	1427s(9.29)	2749s	3.85x
		IST	2126s (6.24)	1412s(9.39)	715s	
	chr8	PT	6515s (3.24)	2020s(10.43)	4495s	18.30x
		IST	2219s (9.50)	1973s(10.68)	246s	
	chr5	PT	6490s (5.09)	2982s(**11.08**)	3508s	**21.03x**
		IST	3188s (**10.37**)	3021s(10.94)	167s	

speedup IST

Chr5: 58.46 minutos para 2.78 minutos

CUDAlign 4.0: Escore e alinhamento em várias GPUs

Alinhamento ótimo do
Cromossomo 1 homem X
Cromossomo 1 chimpanzé



Name	Year	Size	Output	PU	GCUPS
CUDASW++1.0	2009	10,000	score	1 GPU	16.09
CUDASW++2.0	2010	10,000	score	1 GPU	29.70
CUDAlign1.0	2010	10,000,000	score	1 GPU	20.37
CUDAlign2.0	2011	10,000,000	score,align	1 GPU	23.63
CUDAlign2.1	2013	10,000,000	score,align	1 GPU	58.21
CUDASW++3.0	2013	1,000	score	2 GPU+ 4 CPU	185.60
SW#	2013	10,000,000	score,align	2 GPU	65.20
SWAPHI	2014	10,000,000	score	4 PHIs	111.40
SW-Rivyera	2013	1,000	score	128 FPGAs	3040.00
SW-MVM	2014	100	score,align	128 CPUs	900.00
CUDAlign3.0	2014	100,000,000	score	64 GPUs	1726.47
This work	2015	100,000,000	score,align	384 GPUs	10370.00

GCUPS: Billions of Matrix Cells Updated per Second

2838 IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 27, NO. 10, OCTOBER 2016

CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-Wide Alignment in GPU Clusters

Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo, *Senior Member, IEEE*

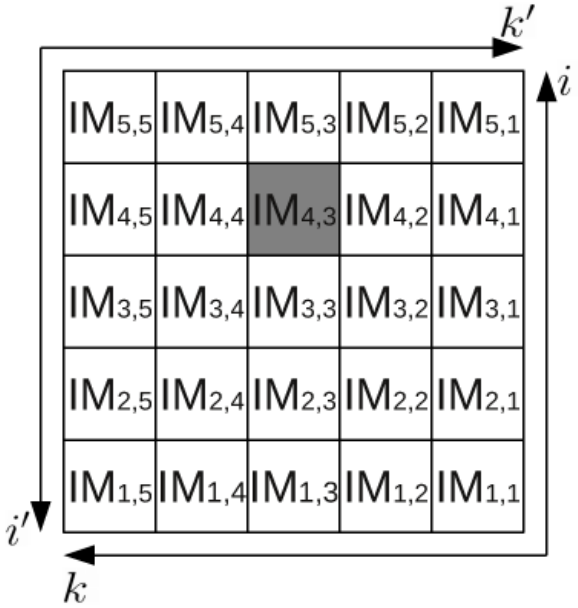
Publicação:
IEEE Transactions Par Dist Syst 2016 – A1

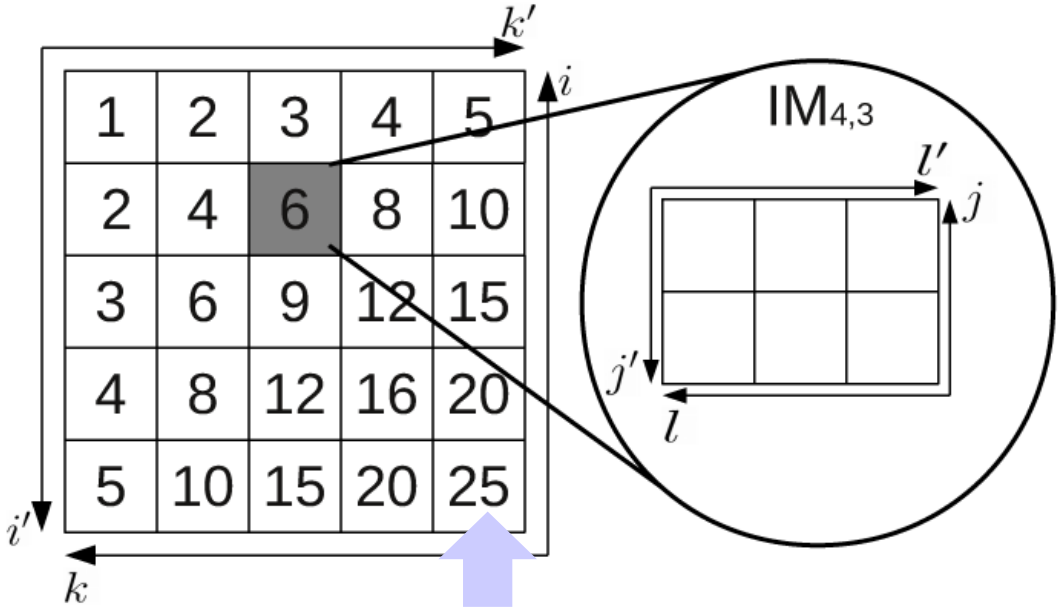
Agenda

- Introdução e Motivação
- Programação Paralela: Visão Geral
- Comparação de Sequências Biológicas
- Tarefas dependentes com padrão wavefront
  - Estudo de caso em FPGA: DIALIGN-Align
  - Estudo de caso em GPU: CUDAlign
- Tarefas dependentes com padrão wavefront aninhado
  - Estudo de caso em GPU: CUDA-Sankoff
- Conclusões e Trabalhos Futuros

CUDA-Sankoff
Matrizes Aninhadas

External Matrix (EM)

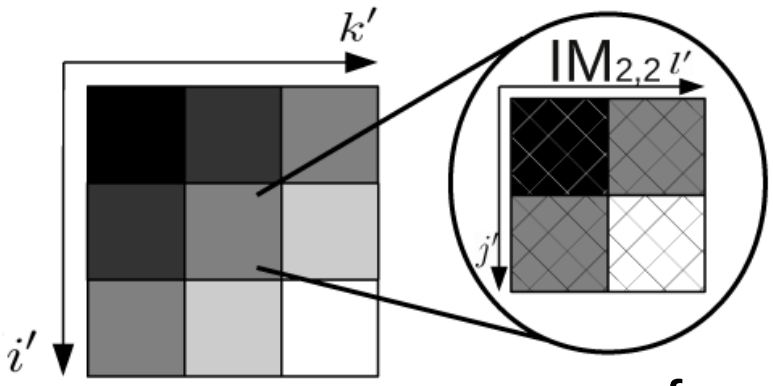




tamanho de
cada matriz interna

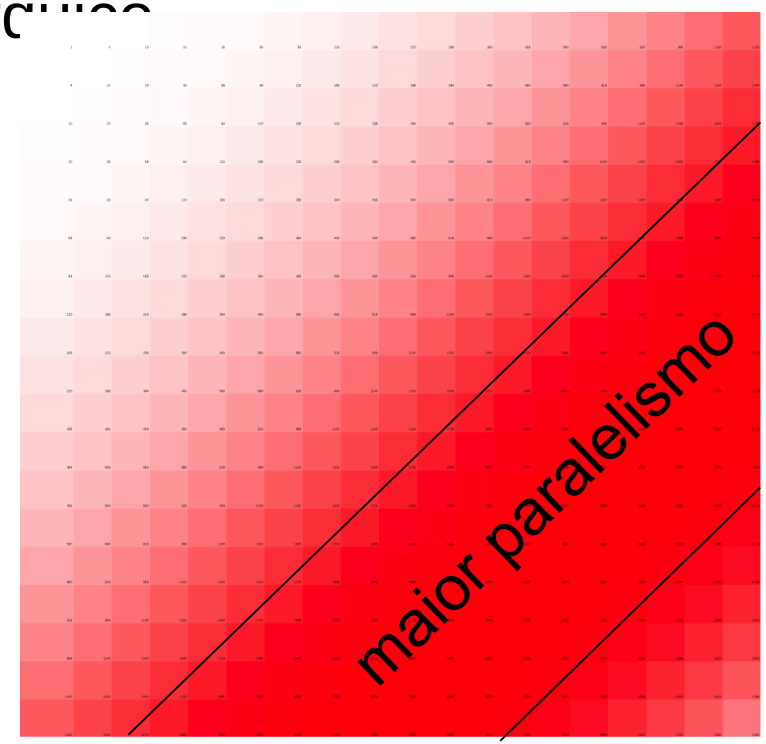
CUDA-Sankoff

- Primeira vez que o algoritmo Sankoff é implementado em GPU
- Decisões:
  - Linearizar a matriz de 4D (i,j,k,l) para simplificar o acesso
  - Paralelismo em wavefront hierárquico

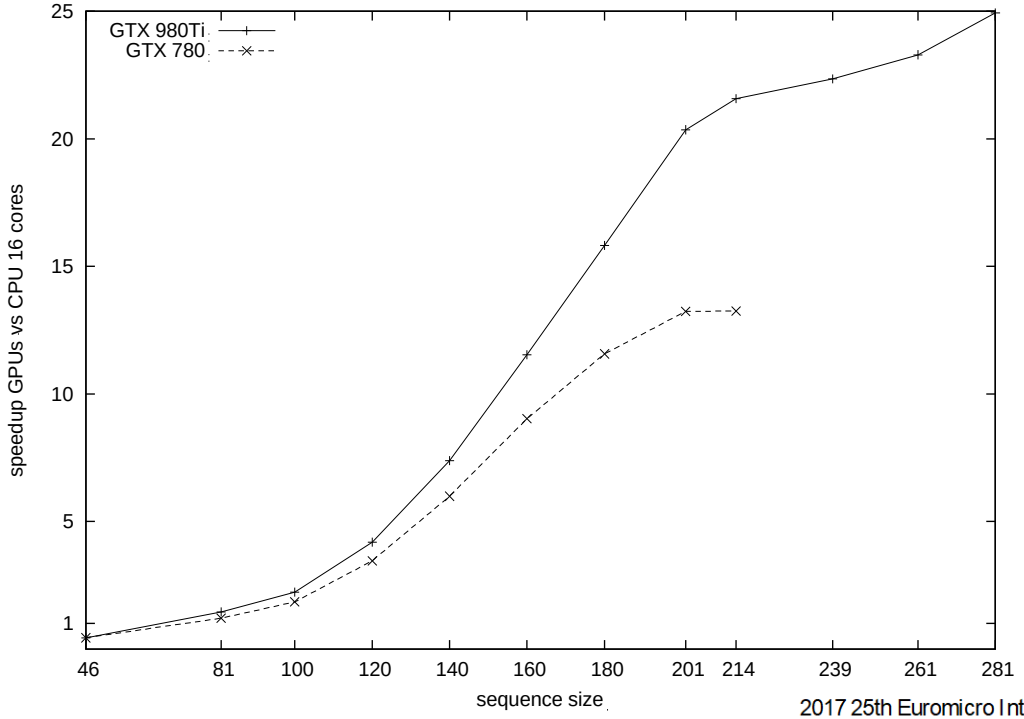


wavefront externo

wavefront interno



CUDA-Sankoff Resultados



Speedup	
1CPU	16 CPU
2.28x	0.43x
10.90x	1.45x
19.11x	2.23x
32.67x	4.19x
45.15x	7.39x
58.50x	11.53x
77.91x	15.82x
91.03x	20.36x
103.19x	21.88x
109.79x	22.35x
108.40x	23.30x
108.69x	24.13x

2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing

Publicação:
Conferência EuroMicro PDP 2017 – A2

CUDA-Sankoff: using GPU to accelerate the pairwise structural RNA alignment

Daniel Sundfeld[□], Jakob H. Havgaard[†], Jan Gorodkin[†], Alba C. M. A. de Melo[□]
[□]Department of Computer Science,
University of Brasilia, Brasilia, DF, Brazil
Email: {sund,alves}@unb.br
[†]Center for non-coding RNA in Technology and Health,
IKVH, University of Copenhagen, Denmark
Email: {hull,gorodkin}@rth.dk

Conclusão

- A programação paralela é fundamental na era do multicore e dos aceleradores.
- Um bom programador tem que:
  - conhecer os fundamentos;
  - ser criativo de maneira a aplicá-los de maneira a extrair o máximo paralelismo;
  - Ser persistente
- Atualmente, a maioria dos sistemas HPC é híbrido e, portanto, a programação de aceleradores (GPU, FPGA, Intel Phi) é necessária.

Conclusão

- Com a programação paralela e hardware adequados, conseguimos fazer cálculos que até então eram considerados impraticáveis (unfeasible):
  - CUDAlign: comparação dos cromossomos 1 homólogos do homem (249 milhões de caracteres) e do chimpanzé (232 milhões de caracteres), com uma matriz de 60 Petacélulas, em duas horas com 384 GPUs.
  - Implementação do algoritmo DIALIGN em FPGA, com speedups de até 140x em comparação com a implementação serial.
  - Implementação do algoritmo Sankoff em GPU com speedups de até 24x em comparação com a implementação multicore.

Agradecimentos

- Alunos:
  - Marcelo Sousa (MsC), Jan M. Correa (PhD), Edans F. O. Sandes (PhD), Daniel Sundfeld (PhD)
- Professores:
  - Ricardo P. Jacobi (UnB, Brazil), George Teodoro (UnB, Brazil), Azzedine Boukerche (UOttawa, Canada), Xavier Martorell (UPC, Espanha), Eduard Ayguade (UPC, Espanha), Jan Gorodkin (UCopenhagen, Dinamarca), Jakob Havgaard (UCopenhagen, Dinamarca)

Marcelo
PackageBLAST

Jan
DIALIGN-Align

Edans
CUDAlign

Daniel
CUDA-Sankoff

Agradecimentos

Barcelona Supercomputing Center (BSC):
Cluster Minotauro – 256 GPUs Tesla M2090
(usamos 64 GPUs no CUDALign 3.0)

Georgia Tech, USA
Keeneland Fullscale System (KFS):
768 GPUs Tesla M2090
(usamos 384 GPUs no CUDALign 4.0)

Muito obrigada!

Alba Cristina M. A. Melo
alves@unb.br

ERAD-SP - 8 de abril de 2017