

Makefile::Parallel

Uma ferramenta para paralelização de processos

Alberto Manuel Brandão Simões
ambs@di.uminho.pt

Trabalho realizado com
Rúben Fonseca e José João Almeida

Simpósio Doutoral da Linguateca 2007b

- processo de alinhamento e de extracção de exemplos demora **demasiado tempo** (uma semana...);
- afinação de parâmetros de algoritmos obriga ao **re-cálculo** de todo o processo;
- esperar uma semana é impensável;
- **paralelizar o processo** ao nível da instrução está fora do âmbito do doutoramento...
- ...no entanto a paralelização **ao nível da aplicação** não é tão dispendiosa...
- ...uma vez que a pipeline já é um conjunto de **pequenos programas**;
- mas é necessário um método expedito de **especificar formalmente dependências** entre as várias aplicações que constituem o processo de alinhamento e extracção de exemplos;



Corpus	PT-EN	PT-ES	PT-FR
Constituição	2 013	2 011	2 013
COMPARA	97 215	—	—
Le Monde Diplomatique	—	—	68 231
JRC	286 008	281 185	277 754
EuroParl	998 830	1 006 895	1 023 841
EurLex	10 394 893	1 111 068	1 710 760

- utilizar uma linguagem conhecida para especificar dependências: `makefile`;
- dependências entre aplicações e não dependência de ficheiros;
- açúcar sintáctico extra para alguns detalhes específicos;
- acções semânticas em Bash (como as makefiles)
- ou em Perl (linguagem de eleição).
- regras paramétricas:
 - instanciar regras em tempo de execução;
 - facilitar paralelismo da mesma aplicação;
 - facilitar execução com parâmetros diferentes;



Cada regra, inclui:

- nome do processo;
- lista de dependências;
- tempo de execução esperado;
- número de processadores necessários;
- acção: descrição do processo.

Example

```
initmat: codify (20:00:00) [1]  
    nat-initmat source.crp target.crp mat.in
```



Instanciação de uma variável

```
foo: ...
...
i <- sub{ $nr = 'cat Eu...'; printf("%03d\n",$_) for (1..$nr); }
```

Regra paramétrica

```
initmat$i: codify (20:00:00)
initmat EurLex/source.$i.crp EurLex/target.$i.crp EurLex/mat.$i.in
```

Regra expandida

```
initmat001: codify (20:00:00)
initmat EurLex/source.001.crp EurLex/target.001.crp EurLex/mat.001.in

initmat002: codify (20:00:00)
initmat EurLex/source.002.crp EurLex/target.002.crp EurLex/mat.002.in

initmat003: codify (20:00:00)
initmat EurLex/source.003.crp EurLex/target.003.crp EurLex/mat.003.in

...
```



Instanciação de uma variável

```
foo: ...  
...  
i <- sub{ $nr = 'cat Eu...'; printf("%03d\n",$_) for (1..$nr); }
```

Regra paramétrica

```
initmat$i: codify (20:00:00)  
    initmat EurLex/source.$i.crp EurLex/target.$i.crp EurLex/mat.$i.in
```

Regra expandida

```
initmat001: codify (20:00:00)  
    initmat EurLex/source.001.crp EurLex/target.001.crp EurLex/mat.001.in  
  
initmat002: codify (20:00:00)  
    initmat EurLex/source.002.crp EurLex/target.002.crp EurLex/mat.002.in  
  
initmat003: codify (20:00:00)  
    initmat EurLex/source.003.crp EurLex/target.003.crp EurLex/mat.003.in  
...
```

Instanciação de uma variável

```
foo: ...  
...  
i <- sub{ $nr = 'cat Eu...'; printf("%03d\n",$_) for (1..$nr); }
```

Regra paramétrica

```
initmat$i: codify (20:00:00)  
    initmat EurLex/source.$i.crp EurLex/target.$i.crp EurLex/mat.$i.in
```

Regra expandida

```
initmat001: codify (20:00:00)  
    initmat EurLex/source.001.crp EurLex/target.001.crp EurLex/mat.001.in  
  
initmat002: codify (20:00:00)  
    initmat EurLex/source.002.crp EurLex/target.002.crp EurLex/mat.002.in  
  
initmat003: codify (20:00:00)  
    initmat EurLex/source.003.crp EurLex/target.003.crp EurLex/mat.003.in  
...
```

```

codify: (20:00:00)
    nat-codify -id=EurLex -tmx eurlex.tmx
    i <- sub{ $nr = 'cat EurLex/nat.cnf |grep nr-chunks|cut -f 2 -d "="; printf("%03d\n",$_) for (1..$nr)
    j <- sub{ print("$_\n") for (2..4); }

ngramsA$j: codify (20:00:00)
    for a in @i; do nat-ngrams -n $j EurLex/source.${a}.crp /tmp/source.$j.ngrams; done
    mv /tmp/source.$j.ngrams EurLex/source.$j.ngrams

ngramsB$j: codify (20:00:00)
    for a in @i; do nat-ngrams -n $j EurLex/target.${a}.crp /tmp/target.$j.ngrams; done
    mv /tmp/target.$j.ngrams EurLex/target.$j.ngrams

ngramsA: ngramsA$j (20:00:00)
    nat-joinGrams -tmp=/tmp -s EurLex

ngramsB: ngramsB$j (20:00:00)
    nat-joinGrams -tmp=/tmp -t EurLex

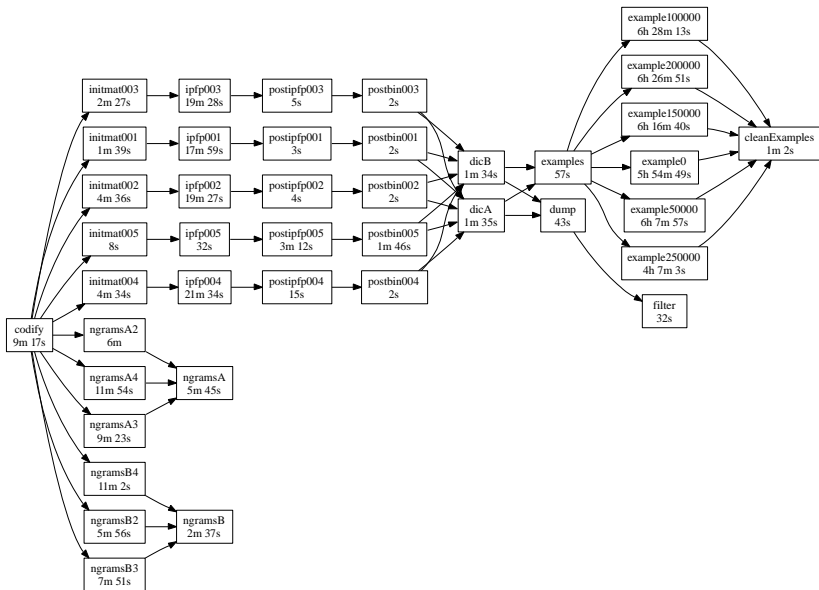
ngrams: ngramsA ngramsB (20:00:00)
    echo 'n-grams=1' >> EurLex/nat.cnf

initmat$i: codify (20:00:00)
    nat-initmat EurLex/source.$i.crp EurLex/target.$i.crp EurLex/mat.$i.in

ipfp$i: initmat$i (20:00:00)
    nat-ipfp 5 EurLex/source.$i.crp EurLex/target.$i.crp EurLex/mat.$i.in EurLex/mat.$i.out
    rm -f EurLex/mat.$i.in

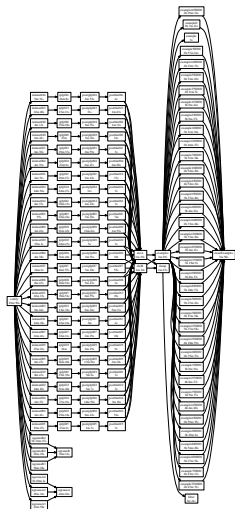
```

Paralelização da Extracção de Exemplos



Na verdade, os processos típicos que têm sido executados, compreendem:

- mais de 150 trabalhos (**mais de 1000 para EurLex**);
- mais de 25 níveis de paralelismo (**mais de 100 para EurLex**);
- mais de 16 milhões de exemplos extraídos;
- mais de 1 milhão de nominais;
- mais de 10 GB de informação produzida;
- tempo crítico de execução: 4h 30m
- **bottleneck**: acesso a disco





- **Computação paralela...**
 - não é só ao nível da instrução;
 - é simples partir processos em processos mais pequenos;
 - se estes processos mais pequenos forem independentes, pode-se tirar partido de paralelismo;
- **Para despoletar processos paralelos...**
 - manualmente é uma dor de cabeça;
 - escrever uma script é uma solução... para poucos processos;
 - o aumento de escala leva à necessidade de uma linguagem de especificação;
- **Linguagem de especificação que...**
 - é orientada ao processo;
 - é genérica (está a ser utilizada por físicos da UM);
 - suporta regras dinâmicas (paramétricas);