Compiladores

Wolfram Language

CEFSA - FTT - EC

GitHub: https://github.com/danielscarvalho/FTT-Benchmark-Java-vs.-C

Linguagem interpretada com ANABOLIZANTES

- Processamento interpretado
- Processamento compilado
- Processamento paralelo, um kernel em cada thread do processador
- Processamento em nuvem
- Processamento em GPU (Placa gráfica - PC Gamer)

Cálculo numérico com valores arbitrários (> 32 bits, 64 bits, etc)

In[7]:=
```
Factorial[30]*Factorial[100]
```

Out[7]=
```
24 755 045 541 954 858 455 090 443 437 500 103 413 905 041 434 592 217 067 605 327 114 859 185 ⋱
 639 350 956 033 183 437 599 938 044 085 431 840 484 973 424 910 456 210 235 131 713 625 755 396 ⋱
 041 959 101 138 206 720 000 000 000 000 000 000 000 000 000 000
```

In[9]:=
```
N[Pi, 1000]
```

Out[9]=
```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899⋱
 86280348253421170679821480865132823066470938446095505822317253594081284811174502⋱
 84102701938521105559644622948954930381964428810975665933446128475648233786783165⋱
 27120190914564856692346034861045432664821339360726024914127372458700660631558817⋱
 48815209209628292540917153643678925903600113305305488204665213841469519415116094⋱
 33057270365759591953092186117381932611793105118548074462379962749567351885752724⋱
 89122793818301194912983367336244065664308602139494639522473719070217986094370277⋱
 05392171762931767523846748184676694051320005681271452635608277857713427577896091⋱
 73637178721468440901224953430146549585371050792279689258923542019956112129021960⋱
 86403441815981362977477130996051870721134999999837297804995105973173281609631859⋱
 50244594553469083026425223082533446850352619311881710100031378387528865875332083⋱
 81420617177669147303598253490428755468731159562863882353787593751957781857780532⋱
 17122680661300192787661119590921642019893809525720106480839541000.
```

Não tem erro como em outras linguagens...

In[2]:=
```
.1 + .2
```

Out[2]=
```
0.3
```

```
> python
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:09:58)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> .1 + .2
0.30000000000000004
>>> exit()
> node
Welcome to Node.js v12.18.4.
Type ".help" for more information.
> .1 + .2
0.30000000000000004
>
```

Processamento interpretado

In[46]:=
```
AbsoluteTiming [Total[N[Table[x^2 * Sin[x], {x, 1, 100 000}]]]]
```

Out[46]=
$\{0.349218\ ,\ 9.32534 \times 10^{9}\}$

Processamento compilado

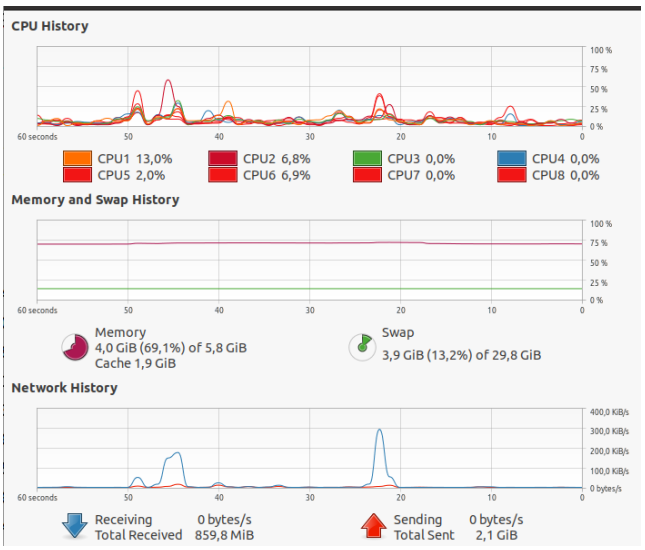
In[47]:=
```
tt = Compile[{}, Total[N[Table[x^2 * Sin[x], {x, 1, 100 000}]]]]
```

Out[47]=
CompiledFunction [WVM Argument count : 0 Argument types : {}]

In[12]:=
```
AbsoluteTiming [tt[]]
```

Out[12]=
$\{0.105559\ ,\ 9.32534 \times 10^{9}\}$

Processamento paralelo na GPU (8 cores)



In[49]:=
```
AbsoluteTiming [Parallelize [Total[N[Table[x^2 * Sin[x], {x, 1, 100 000}]]]]]
```

Parallelize : Total [N[Table [x^2 Sin [x], {x, 1, 100000 }]]] cannot be parallelized ; proceeding with sequential evaluation .

Out[49]= $\{0.351341 , 9.32534 \times 10^9\}$

Nem todo programa pode ser paralelizado... como sistemas dinâmicos

Processando remotamente na nuvem da Wolfram

In[20]:=
```
CloudSubmit [
 CloudPut[AbsoluteTiming [Total[N[Table[x^2 * Sin[x], {x, 1, 100 000}]]]], "ftt-task"]]
```

Out[20]= TaskObject[

Task UUID : 43b670d9 -fa90 -4a84 -a4dc -3a2d0fe02e33
Task environment : Cloud
Task type : Cloud
Evaluation expression : CloudPut [AbsoluteTiming [Total [N[Table [x^2 Sin [x], {≪1≫}]]]], ...]

]

In[22]:=
```
CloudGet["ftt-task"]
```

Out[22]= $\{0.262478 , 9.32534 \times 10^9\}$

Processamento batch na AWS (HPC)

Processamento em GPU (CUDA)

In[ ]:=
```
Needs["CUDALink`"]
```

This generates a random list of reals:

In[ ]:=
```
lst = RandomReal[1., {100 000}];
```

This computes the one-dimensional Fourier transform using CUDA:

In[ ]:=
```
AbsoluteTiming [CUDAFourier [lst]]
```

Out[ ]=
```
{0.020023 , {0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i,
   0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i, ··· 99 983 ···, 0. + 0. i,
   0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i, 0. + 0. i}}
```

large output | show less | show more | show all | set size limit...

The result agrees with the Wolfram Language by CPU:

In[ ]:=
```
AbsoluteTiming [Fourier [lst]]
```

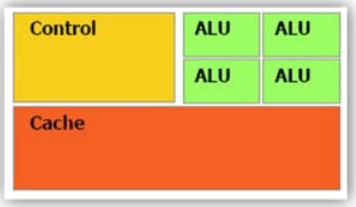
Out[ ]=
```
{0.105134 , {158.528 - 5.05999 × 10^-17 i, -0.117935 - 0.14171 i,
   ··· 99 996 ···, -0.0702539 - 0.145001 i, -0.117935 + 0.14171 i}}
```

large output | show less | show more | show all | set size limit...

CUDA -

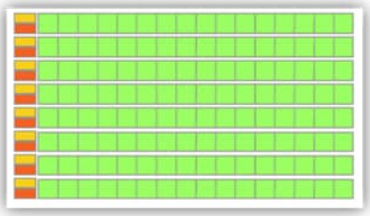
GPU:

CPU



* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU



* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore