

**Mestrado em Engenharia Informática**

*Algoritmos Paralelos - 2014/2015*

MapReduce com MPI

18 de Junho de 2015

**Fábio Gomes** pg27752

# **Índice**

[**Índice** 2](#_Toc422436496)

[Introdução 3](#_Toc422436497)

[Funcionamento do Algoritmo 4](#_Toc422436498)

[Versão MPI simples 5](#_Toc422436499)

[Versão MapReduceMPI 6](#_Toc422436500)

[Tempos e Speedup 8](#_Toc422436501)

[*Tabela Comparativa* 8](#_Toc422436502)

[Conclusão e Análise de Resultados 10](#_Toc422436503)

# Introdução

O problema que nos foi apresentado trata-se de implementar 2 versões do Friendly Numbers em Map Reduce com MPI e com MRMPI. Depois comparar os resultados.

# 

# Funcionamento do Algoritmo

Tendo como objetivo encontrar os números amigáveis de start a end, o algoritmo original está em OpenMP para que se possa tirar partido do paralelismo. Para comparar se 2 números são amigáveis, têm que ter o numerador e o denominador iguais, por isso temos que os guardar para futura comparação. Portanto é guardado um array num, den e the\_num para sabermos de que números se tratam.

Cada threads para o seu número guarda nos arrays os valores calculados. No fim é feita uma comparação um a um para encontrar pares num/den iguais.

void FriendlyNumbers **(**int start**,** int end**)**

**{**

int last **=** end**-**start**+**1**;**

int **\***the\_num **=** **new** int**[**last**];**

int **\***num **=** **new** int**[**last**];**

int **\***den **=** **new** int**[**last**];**

#pragma omp parallel

**{** int i**,** j**,** factor**,** ii**,** sum**,** done**,** n**;**

// -- MAP --

#pragma omp for schedule (dynamic, 16)

**for** **(**i **=** start**;** i **<=** end**;** i**++)** **{**

ii **=** i **-** start**;**

sum **=** 1 **+** i**;**

the\_num**[**ii**]** **=** i**;**

done **=** i**;**

factor **=** 2**;**

**while** **(**factor **<** done**)** **{**

**if** **((**i **%** factor**)** **==** 0**)** **{**

sum **+=** **(**factor **+** **(**i**/**factor**));**

**if** **((**done **=** i**/**factor**)** **==** factor**)** sum **-=** factor**;**

**}**

factor**++;**

**}**

num**[**ii**]** **=** sum**;** den**[**ii**]** **=** i**;**

n **=** gcd**(**num**[**ii**],** den**[**ii**]);**

num**[**ii**]** **/=** n**;**

den**[**ii**]** **/=** n**;**

**}** // end for

// -- REDUCE --

#pragma omp for schedule (static, 8)

**for** **(**i **=** 0**;** i **<** last**;** i**++)** **{**

**for** **(**j **=** i**+**1**;** j**<** last**;** j**++)** **{**

**if** **((**num**[**i**]** **==** num**[**j**])** **&&** **(**den**[**i**]** **==** den**[**j**]))**

printf **(**"%d and %d are FRIENDLY \n"**,** the\_num**[**i**],** the\_num**[**j**]);**

**}**

**}**

**}** // end parallel region

**delete[]** the\_num**;**

**delete[]** num**;**

**delete[]** den**;**

**}**

# Versão MPI simples

A Paralelização com MPI é feita dividindo um range de valores para cada processo. Depois invocar a FriendlyNumbers com esse range e no fim enviar para o processo 0 todos os dados calculados, portanto o pid faz também trabalho computacional mas irá depois receber a informação de todos e calcular os números amigáveis.

int main**(**int argc**,** char **\*\***argv**)**

**{**

unsigned int start **=** atoi**(**argv**[**1**]),** end **=** atoi**(**argv**[**2**]);**

MPI\_Init**(&**argc**,&**argv**);**

double time**=**MPI\_Wtime**();** int pid**,**n\_proc**;**

MPI\_Status status1**,**status2**;**

MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,&**n\_proc**);** MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,&**pid**);**

size **=** end**-**start**+**1**;**

int offset **=** size **/** **(**n\_proc**);**

int off **=** 0**,**proc**;**

printf**(**"pid:%d start:%d end:%d offset:%d size:%d\n"**,**pid**,**start**+(**offset**\***pid**),** start**+(**offset**\*(**pid**+**1**)),**offset**,**size**);**

**if(**pid**==**0**){**

num **=** **new** int**[**size**];** den **=** **new** int**[**size**];}**

**else**

**{**

num **=** **new** int**[**offset**];** den **=** **new** int**[**offset**];}**

FriendlyNumbers**(**start**+(**offset**\*(**pid**)),** start**+(**offset**\*(**pid**+**1**)-**1**));**

**if(**pid**==**0**){**

int i**=**1**,**j**;**

**for(**i**;**i**<**n\_proc**;**i**++){**

MPI\_Recv**(&**num**[(**i**)\***offset**],**offset**,** MPI\_INT**,** i**,**1**,**MPI\_COMM\_WORLD**,&**status1**);**

MPI\_Recv**(&**den**[(**i**)\***offset**],**offset**,** MPI\_INT**,** i**,**2**,**MPI\_COMM\_WORLD**,&**status2**);** **}**

// -- REDUCE --

#pragma parallel omp for schedule (static, 8)

**for** **(**i **=** 0**;** i **<** size**;** i**++)**

**for** **(**j **=** i**+**1**;** j**<** size**;** j**++)**

**if** **((**num**[**i**]** **==** num**[**j**])** **&&** **(**den**[**i**]** **==** den**[**j**]))**

printf **(**"%d and %d are FRIENDLY \n"**,** start**+**i**,** start**+**j**);**

time**=**MPI\_Wtime**()-**time**;**

printf**(**"Time:%f seconds\n"**);**

**}**

**else{**

MPI\_Send**(&**num**[**0**],**offset**,** MPI\_INT**,** 0**,**1**,**MPI\_COMM\_WORLD**);**

MPI\_Send**(&**den**[**0**],**offset**,** MPI\_INT**,** 0**,**2**,**MPI\_COMM\_WORLD**);**

**}**

MPI\_Finalize**();**

**}**

# Versão MapReduceMPI

Foi necessário introduzir a *struct* Pair que vai ser a Chave do *Map*, é constituída por 2 inteiros, que serão correspondidos ao numerador e denominador. A charTointStar vai passar um array de *char’s* para inteiros para podermos ler os números amigáveis, correspondentes ao Valor do *Map*. A *output* trata do *reduce*, pega nas chaves que tenham mais que um valor (ou seja têm pelo menos um par de números amigáveis) e faz o print dos números.

**typedef** struct sPair **{**

int x**,** y**;**

**}**Pair**;**

void charTointStar**(**int**\*** ints**,** char**\***multivalue**,** int nvalues**,**int**\*** vb**)**

**{**

int**\*** values **=** **(**int**\*)** multivalue**;**

**for** **(**int i **=** 0**;** i **<** nvalues**;** **++**i**)**

**{**

ints **[**i**]** **=** values**[**i**];**

**}**

**}**

void output**(**char **\***key**,** int keybytes**,** char **\***multivalue**,** int nvalues**,** int **\***valuebytes**,** KeyValue **\***kv**,** void **\***ptr**)**

**{**

int**\*** numbers**=(**int**\*)**malloc**(sizeof(**int**)\***nvalues**);**

charTointStar**(**numbers**,**multivalue**,**nvalues**,**valuebytes**);**

**if(**nvalues**>**1**)**

**{**

**for** **(**int i **=** 0**;** i **<** nvalues**;** i**++)**

**{**

printf**(**"%d "**,** multivalue**[**i**]);**

**}**

printf**(**"\n"**);**

**}**

**}**

A FriendlyNumbers é a função que trata do *Map*. É criado o *Pair p* que será a chave e adicionado o Valor com o número atual.

void FriendlyNumbers**(**int itask**,** KeyValue **\***kv**,** void**\*** ptr**)**

**{**

int num**,**den**;**

#pragma omp parallel

**{** int i**,** j**,** factor**,** ii**,** sum**,** done**,** n**;**

// -- MAP --

#pragma omp for schedule (dynamic, 16)

**for** **(**i **=** start**;** i **<=** end**;** i**++)** **{**

ii **=** i **-** start**;**

sum **=** 1 **+** i**;**

done **=** i**;**

factor **=** 2**;**

**while** **(**factor **<** done**)** **{**

**if** **((**i **%** factor**)** **==** 0**)** **{**

sum **+=** **(**factor **+** **(**i**/**factor**));**

**if** **((**done **=** i**/**factor**)** **==** factor**)** sum **-=** factor**;**

**}**

factor**++;**

**}**

num **=** sum**;**

den **=** i**;**

n **=** gcd**(**num**,** den**);**

num **/=** n**;**

den **/=** n**;**

Pair p**;**p**.**x**=**num**;**p**.**y**=**den**;**

int val **=** i**;**

kv**->**add**((**char**\*)&**p**,sizeof(**Pair**),(**char**\*)** **&**val**,sizeof(**int**));**

**}** // end for

**}**

// end parallel region

**}**

No main, instancia-se o MapReduce, vem a barreira, é iniciado o map com o FriendlyNumbers, é feito o *collate* e de seguida o *reduce* com o output.

// MapReduce

MapReduce **\***mr **=** **new** MapReduce**(**MPI\_COMM\_WORLD**);**

mr**->**verbosity**=**0**;**

mr**->**timer **=** 1**;**

MPI\_Barrier**(**MPI\_COMM\_WORLD**);**

int nNumber **=** mr**->**map**(**n\_proc**,&**FriendlyNumbers**,NULL);**

int nChunks **=** mr**->**mapfilecount**;**

mr**->**collate**(NULL);** // Collate Keys

**if(**pid**==**0**)**

printf**(**"The following numbers are friendly:\n"**);**

int nunique **=** mr**->**reduce**(**output**,NULL);**

MPI\_Barrier**(**MPI\_COMM\_WORLD**);**

# Tempos e Speedup

Foram Realizados 5 testes para cada *Size* e número de Processos, calculada a média desses tempos e feito o *Speedup* baseado no tempo da versão MPI.

## *Tabela Comparativa*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Size | | | | Speedups | | | |
|  |  |  | 1000 | 10000 | 1000000 | 10000000 | 1000 | 10000 | 1000000 | 10000000 |
| Procs | 2 | MPI | 0,000648 | 0,031002 | 2,361124 | 192,8277 | 48,81481 | 2,027772 | 1,026414 | 1,000801 |
| MRMPI | 0,031632 | 0,062865 | 2,423491 | 192,9821 |
| 4 | MPI | 0,000367 | 0,018001 | 1,359308 | 111,2274 | 110,7711 | 3,269429 | 1,039153 | 1,002239 |
| MRMPI | 0,040653 | 0,058853 | 1,412529 | 111,4764 |
| 8 | MPI | 0,0008 | 0,009842 | 0,730489 | 59,52372 | 89,80625 | 8,273217 | 1,116176 | 1,004068 |
| MRMPI | 0,071845 | 0,081425 | 0,815354 | 59,76585 |
| 10 | MPI | 0,000709 | 0,008303 | 0,589432 | 47,96692 | 106,2003 | 9,78297 | 1,140802 | 1,004857 |
| MRMPI | 0,075296 | 0,081228 | 0,672425 | 48,19988 |
| 12 | MPI | 0,041271 | 0,047512 | 0,535457 | 40,31815 | 2,855274 | 2,636155 | 1,154401 | 1,004971 |
| MRMPI | 0,11784 | 0,125249 | 0,618132 | 40,51857 |
| 16 | MPI | 0,00182 | 0,005867 | 0,375138 | 30,62136 | 45,32912 | 14,77808 | 1,256407 | 1,015962 |
| MRMPI | 0,082499 | 0,086703 | 0,471326 | 31,11015 |
| 20 | MPI | 0,001776 | 0,00462 | 0,302531 | 24,51627 | 49,09685 | 19,35844 | 1,313492 | 1,055852 |
| MRMPI | 0,087196 | 0,089436 | 0,397372 | 25,88555 |

A versão em MPI foi sempre melhor que a MRMPI mas com o aumento do Size a diferença diluiu-se.

## *mpiP para versão MPI*

Como podemos analisar pelo resultado obtido da execução com *mpiP*, quanto maior o *Size* maior o tempo que é preciso esperar para receber os valores no primeiro nó, porque o cálculo do maior divisor comum é mais lento para valores altos e o *pid 0* é o que acaba mais rápido. Para valores baixos a diferença ainda é pouca. Para 4 processos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Call** | **Site** | **Time** | **App%** | **MPI%** | **COV** |
| **1000** | **Recv** | 1 | 0.016 | 1.06 | 2,09 | 0.00 |
| **Recv** | 2 | 0.337 | 22.24 | 44,11 | 0.00 |
| **Send** | 3 | 0.024 | 1.58 | 3,14 | 0.57 |
| **Send** | 4 | 0.387 | 25.54 | 50,65 | 0.17 |
| **100000** | **Recv** | 1 | 0.353 | 0.01 | 0,03 | 0.00 |
| **Recv** | 2 | 1.12e+03 | 25.75 | 99,85 | 0.00 |
| **Send** | 3 | 0.344 | 0.01 | 0,03 | 0.27 |
| **Send** | 4 | 0.936 | 0.02 | 0,08 | 0.13 |

## *mpiP para versão MRMPI*

Era de esperar que para valores baixos a Barreira introduzida para sincronismo fosse trazer problemas e é o que o *mpiP* confirmou. Depois como seria de esperar a Allreduce tem muita carga computacional, o mrMPI tem que enviar para todos os processos os valores para fazerem reduce. Para 4 processos.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Call** | **Site** | **Time** | **App%** | **MPI%** | **COV** |  |  | **Call** | **Site** | **Time** | **App%** | **MPI%** | **COV** |
| **1000** | Barrier | 3 | 0,803 | 0,5 | 14,19 | 0,2 |  | **100000** | Allreduce | 16 | 1,12E+03 | 19,73 | 50,42 | 0 |
| Allreduce | 50 | 0,405 | 0,25 | 7,16 | 0 |  | Allreduce | 50 | 725 | 12,77 | 32,63 | 0 |
| Allreduce | 25 | 0,328 | 0,21 | 5,8 | 0 |  | Allreduce | 33 | 356 | 6,26 | 16 | 0 |
| Allreduce | 7 | 0,288 | 0,18 | 5,09 | 0 |  | Alltoallv | 63 | 3,14 | 0,06 | 0,14 | 0 |
| Allreduce | 16 | 0,272 | 0,17 | 4,81 | 0 |  | Alltoallv | 46 | 3,1 | 0,05 | 0,14 | 0 |
| Allreduce | 67 | 0,256 | 0,16 | 4,52 | 0 |  | Alltoallv | 29 | 2,64 | 0,05 | 0,12 | 0 |
| Allreduce | 42 | 0,217 | 0,14 | 3,84 | 0 |  | Alltoallv | 11 | 1,73 | 0,03 | 0,08 | 0 |
| Allreduce | 20 | 0,209 | 0,13 | 3,69 | 0 |  | Allreduce | 64 | 1,69 | 0,03 | 0,08 | 0 |
| Barrier | 56 | 0,142 | 0,09 | 2,51 | 0 |  | Allreduce | 47 | 1,66 | 0,03 | 0,07 | 0 |
| Barrier | 39 | 0,142 | 0,09 | 2,51 | 0 |  | Allreduce | 30 | 1,64 | 0,03 | 0,07 | 0 |
| Barrier | 22 | 0,135 | 0,08 | 2,39 | 0 |  | Barrier | 3 | 0,715 | 0,01 | 0,03 | 0,08 |
| Allreduce | 69 | 0,132 | 0,08 | 2,33 | 0 |  | Allreduce | 53 | 0,598 | 0,01 | 0,03 | 0 |
| Allreduce | 33 | 0,12 | 0,08 | 2,12 | 0 |  | Allreduce | 70 | 0,566 | 0,01 | 0,03 | 0 |
| Barrier | 31 | 0,112 | 0,07 | 1,98 | 0 |  | Allreduce | 36 | 0,462 | 0,01 | 0,02 | 0 |
| Allreduce | 52 | 0,102 | 0,06 | 1,8 | 0 |  | Allreduce | 18 | 0,309 | 0,01 | 0,01 | 0 |
| Alltoall | 37 | 0,096 | 0,06 | 1,7 | 0 |  | Barrier | 13 | 0,235 | 0 | 0,01 | 0 |
| Alltoall | 54 | 0,096 | 0,06 | 1,7 | 0 |  | Allreduce | 7 | 0,228 | 0 | 0,01 | 0 |
| Alltoall | 21 | 0,094 | 0,06 | 1,66 | 0 |  | Allreduce | 59 | 0,179 | 0 | 0,01 | 0 |
| Alltoall | 2 | 0,093 | 0,06 | 1,64 | 0 |  | Allreduce | 1 | 0,169 | 0 | 0,01 | 0 |
| Allreduce | 55 | 0,089 | 0,06 | 1,57 | 0 |  | Allreduce | 55 | 0,133 | 0 | 0,01 | 0 |

# Conclusão e Análise de Resultados

Os valores, obtidos no *compute-321-1* com job de 20 *processos*, de Speedup são muito semelhantes à medida que o Size aumenta. Para valores baixos a utilização do MapReduce não é recomendável.