Nombre d'Or (Golden Ratio), Mégadonnées (Big Data) et Grands Nombres

Tayeb Lemlouma
Tayeb.Lemlouma@irisa.fr
Février 2015 (*version draft*).

Résumé

Avez vous essayé (ou eu l'occasion) de travailler avec des grandes quantités d'informations dans votre travail, études ou recherches ? Si ce n'est pas le cas, cet article, destiné au grand public, peut vous donner un petit aperçu.

1 Introduction

Les « mégadonnées » (ou big data) représentent des données massives qui sont très volumineuses et par conséquent très difficiles à traiter. Ce traitement dépend du domaine où sont impliquées ces données. On peut vouloir les visualiser, les analyser, en extraire des mesures, faire des comparaisons ou des explorations, les stocker si nécessaire... On retrouve les « big data » partout et depuis quelques années cela devient un domaine prioritaire de recherche qui revient au premier plan quelles que soient les spécialités : les réseaux énergétiques, l'Internet et l'Internet des objets, l'intelligence artificielle, la sécurité, la biologie, l'écologie, la santé et l'e-santé... Ceci est sans parler des domaines classiques où l'on a toujours eu tendance à manipuler ces grandeurs, tel qu'en physique quantique avec l'infiniment petit et l'infiniment grand qui sont mathématiquement liés par nature. Par exemple, le CERN (l'organisation européenne pour la recherche nucléaire) a annoncé, en 2013, qu'elle avait franchi le seuil de 100 pétaoctets de données informatiques, soit un million de million d'octets : l'équivalent d'environ 700 ans de films en haute définition [1] ! Les calculs intensifs ne se retrouvent pas uniquement chez les physiciens : il a fallu traiter un pétaoctet pour le film Avatar ; Google traitait, déjà en 2009, 24 pétaoctets par jour et les exemples vont se multiplier d'une manière exponentielle et vont toucher de plus en plus du monde ! Ce qui est sûr, c'est que nos méthodes et ressources classiques -telles qu'elles sont actuellement- restent impuissantes et les experts s'accordent que le big data est l'un des grands défis informatiques de la prochaine décennie.

On a rarement l'occasion de tester nos ordinateurs pour des calculs intensifs, de traitement de grandes masses de données ou d'impliquer dans nos calculs de très grands nombres. Mais une fois qu'on le fait, on s'aperçoit de pas mal de mystères de nos toutes puissantes machines qui affichent soudainement de grosses limites que nous serons obligés de contourner. Par exemple, avez vous essayé sous Matlab [1] d'afficher une chaine (ou un nombre) contenant plus de

1. Matlab est l'un des fameux outils informatiques utilisés par les mathématiciens et les scientifiques pour effectuer des programmes de calcul

25,000 caractères (chiffres) ? Si vous ne l'avez pas fait, je vous préviens que le résultat va se solder par l'erreur suivante : « *Output truncated. Text exceeds maximum line length of 25 000 characters for Command Window display* ». Afficher plus de 25,000 caractères sur une même ligne n'est pas prévu !

Je note qu'une fois que l'on goute aux joies des très grands nombres, on devient presque dépendant ! Le calcul des « petits » nombres classiques devient presque ennuyeux ! Evidemment, qui dit très grand nombre dit aussi « très petits nombres » : si x est un très grand nombre eh bien $1/x$ est un très petit nombre. Cela est équivalent dans la complexité mais aussi dans les défis.

Nos premières expérimentations avec le big data et les calculs de très grands nombres concernent un travail dans le domaine de l'e-santé [3]. Dans [3] nous avons eu à effectuer une quantité très importante de calculs (une vingtaine de trillions d'évaluations) sans avoir recours aux services d'un centre de calcul nucléaire ! Le travail avec les très grands nombres dans [3] avait comme objectif de simuler toutes les combinaisons (évaluations) possibles des niveaux de dépendance (besoin d'aides) des personnes afin de comparer les modèles d'évaluation des personnes dépendantes dans toutes les facettes. Ce travail continue et engendre encore du calcul intensif avec la simulation de flux de données issues d'un ensemble de capteurs utilisés dans un habitat intelligent et pour une longue période de capture (surveillance de personnes dépendantes).

Cet article destiné au grand public a été écrit suite à une discussion rapide sur les théories de construction des pyramides (rien à voir avec nos compétences !). Ces théories m'ont vite mené vers le nombre d'or Φ (que je ne connaissais pas !) mais le défi s'est vite concrétisé par : « comment peut on l'estimer avec un maximum de précisions possibles ?», en tout cas avec la précision que ma petite machine pourra m'assurer ! Ceci était uniquement pour le plaisir de le faire [2] !

2 La Valeur de Φ

Le nombre d'or Φ se définit comme étant la solution de l'équation $x+1 = x^2$ ou de $x - 1 = \frac{1}{x}$. Je ne vais pas aborder tout ce qui se dit par rapport au nombre d'or et de ses mystères qu'on retrouve (ou qu'on cherche à retrouver) partout et dans des domaines très différents. Cela vaut (peut être) le coup de jeter un coup d'œil sur Internet sur ce sujet. En tout cas, à lui seul, ce nombre regroupe les théories les plus farfelues [2] et les applications les plus terre à terre. Prenez (au hasard) son utilisation dans la vision par ordinateur concernant les visages humains dans cet article scientifique de 2013 [4] et ses propriétés amusantes que l'on retrouve sur les proportions du corps humain. Ce qui est marrant, c'est que les anciens (les très anciens !) l'utilisaient avec des moyens de traçage géométrique sans avoir recours à une grosse calculette ou à un ordinateur puissant. On sait tracer un segment d'une longueur choisie, on sait tracer un angle droit avec l'intersection des cercles, on sait tracer la moitié

2. «*Ten decimal places of π are sufficient to give the circumference of the earth to a fraction of an inch, and thirty decimal places would give the circumference of the visible universe to a quantity imperceptible to the most powerful microscope* » Simon Newcomb (1835-1909)

d'un segment (toujours avec l'intersection des cercles) et hop, le nombre d'or est tracé avec sa valeur exacte sur le terrain (Figure 1). En effet, la longueur du trait bleu de la figure 1 vaut $\frac{1}{2}$ (ca') plus $\sqrt{1^2 + (\frac{1}{2})^2}$ (théorème de Pythagore). Ce qui donne après simplification $\frac{1+\sqrt{5}}{2}$ qui est la valeur de Φ.

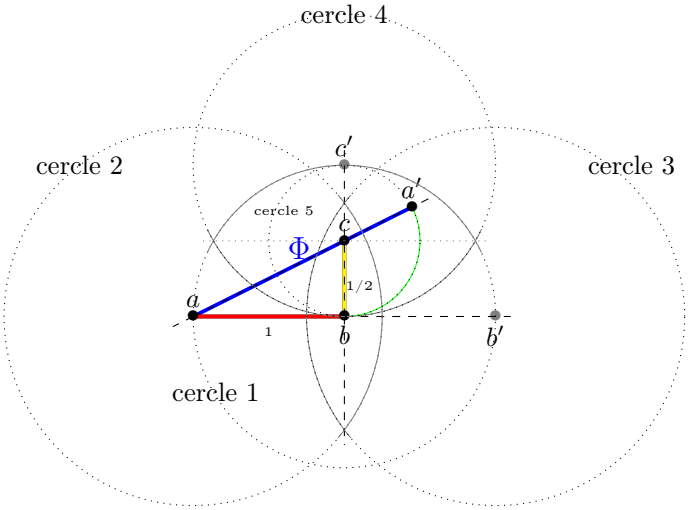

FIGURE 1 – comment tracer une longueur Φ en partant de rien ?

Puisque le nombre d'or Φ se définit comme étant la solution de l'équation $x + 1 = x^2$ (ou de $x - 1 = \frac{1}{x}$). Sa solution positive est donc $\frac{1+\sqrt{5}}{2}$ ($\Delta = b^2 - 4.a.c = 5 \Rightarrow x_1 = \frac{-b+\sqrt{\Delta}}{2a} = \frac{1+\sqrt{5}}{2}$ et $x_2 = \frac{-b-\sqrt{\Delta}}{2a} = \frac{1-\sqrt{5}}{2}$). Donc, on connaît la formulation de sa valeur exacte, rien de mystérieux alors ! Oui, je vous assure, sortez votre calculatrice et calculez le ! Sauf que j'ai oublié de vous dire que la valeur que vous voyez affichée n'est qu'une valeur vulgairement approximative car il y a une infinité de chiffres après la virgule.

Une manière simple de calculer –par programme informatique– une valeur approximative de Φ est d'utiliser une propriété de la suite dite de Fibonacci. Leonardo Fibonacci (connu par son nom d'usage Leonardo *Pisano*) est un mathématicien italien qui a défini une suite de nombres entiers après qu'il s'est attaqué à résoudre un problème de croissance de lapins ! Chaque nouveau terme de la suite Fibonacci est la somme des deux derniers nombres de la suite. On commence par les deux premiers termes : 1 et 1. Le troisième terme est donc 2 (soit 1+1). Le quatrième terme est la somme du troisième et du deuxième terme, c'est à dire 2+1 ce qui donne la valeur 3, ainsi de « suite », car la suite est sans fin. La propriété que nous utiliserons plus tard pour approximer Φ est que « le résultat de la division entre deux termes consécutifs de la suite de Fibonacci représente la meilleure approximation du nombre d'or ».

Ce qui est amusant, c'est que pour un calcul simple –sans approximation– de Φ par ordinateur (c'est à dire si on demande à notre ordinateur de calculer, d'une manière simple, $\frac{1+\sqrt{5}}{2}$), le résultat obtenu va être erroné à partir du 16$^{\text{ième}}$ chiffre

après la virgule ! Si on effectue le même calcul avec une calculatrice, le résultat est erroné à partir du 13ème chiffre après la virgule ! Avec un ordinateur, le calcul de $\frac{1+\sqrt{5}}{2}$ donne 1.6180339887498949025257388711906969547271728515625. Avec une calculatrice, $\frac{1+\sqrt{5}}{2}$ donne 1,6180339887499.

Une manière simple de calculer le degré d'erreur d'une valeur donnée de Φ est de calculer 5 -$(2\Phi - 1)^2$. Cette dernière valeur est sensée être nulle car si $\Phi = \frac{1+\sqrt{5}}{2}$ par conséquent 5 -$(2\Phi - 1)^2$ devrait être nul. Limitons nous dans un premier temps à seulement 49 chiffres après la virgule, ce que donne l'ordinateur -par défaut- lorsqu'on lui demande de calculer $\frac{1+\sqrt{5}}{2}$. Ce résultat de $\frac{1+\sqrt{5}}{2}$ donne une erreur d'ordre -0.00000000000000089 alors qu'avec une approximation Fibonacci avec une suite de seulement 50 termes donne une erreur de 0.000000000000000000096. Cette approximation simple est donc meilleure que ce que notre ordinateur peut calculer. Notez que cette approximation de Fibonacci avec seulement 50 termes de la suite ne demande que quelques petites opérations d'additions (peu d'itérations en terme de programmation informatique) avec un temps de calcul de 0.02 secondes avec un code informatique non optimisé que nous verrons plus tard.

3 Calcul et Estimation

Définissons un algorithme simple pour calculer une estimation du nombre d'or en utilisant, par exemple, la propriété de la suite de Fibonacci [3]. L'Algorithme I effectue des itérations pour calculer les 3,000,000 premiers termes de la suite (lignes 5 à 12) puis calcule une estimation de Φ en divisant le dernier terme obtenu par l'avant dernier terme (ligne 13). Dans les calculs prévus pour de grandes masses de données ou impliquant des très larges nombres, il est conseillé, en tout cas dans la phase de test, de déterminer la durée totale nécessaire à l'achèvement du traitement et d'afficher la progression du calcul afin d'éviter la sensation de blocage de votre algorithme. Bien sûr, si vous voulez optimiser le temps de votre programme, il est inutile de le surcharger avec des affichages gourmands en temps et en mémoire. Dans l'Algorithme I, les lignes 2 et 14 permettent de déterminer le temps total de calcul en faisant la soustraction entre le temps de fin et le temps du début. Les lignes 9 à 11 permettent d'afficher la progression du calcul.

Si vous voulez exécuter l'Algorithme I avec un ordinateur en utilisant un langage de programmation donné, inutile de l'implémenter tel qu'il est ! Bien qu'il soit correct et fonctionnel (pour de petits calculs), le code informatique équivalent à cet algorithme doit être adapté. En effet, les variables utilisées (i, a, b, c et Φ) vont rapidement dépasser les limites mémoires disponibles et prévues pour des variables classiques et l'exécution informatique sera très rapidement bloquée. Elle générera très probablement des erreurs selon le langage de programmation et le système d'exploitation utilisés.

3. Il y a évidemment d'autres méthodes d'approximation de Φ telle avec la méthode de Newton pour estimer $\sqrt{5}$. Le défi est d'utiliser la méthode qui s'adapte bien aux ressources informatiques utilisées (processeur, mémoire...)

Algorithm 1 estimation du nombre d'or avec la suite de Fibonacci

```
1: procedure EstimateGoldenRatio
2:     afficher le temps courant ;
                    Initialisation des deux premiers termes, a et b, de la suite Fibonacci
3:     a ← 1;
4:     b ← 1;
                              Génération de 3 millions de termes de de la suite Fibonacci
5:     for i = 1 → 3,000,000 do
6:         c ← a + b;
7:         a ← b;
8:         b ← c;
9:         if mod (i,100,000) == 0 then
10:            afficher la progression : valeur de i et le temps courant ;
11:        end if
12:    end for
                              Calcul de Φ en divisant les deux derniers termes de la suite
13:    Φ ← b/a;
14:    afficher le temps courant ;
15: end procedure
```

La Figure 2 représente un code Matlab simple [4] qui calcule, en adaptant l'algorithme I, une estimation du nombre Φ à un million de chiffres après la virgule. Grace à notre code informatique, on peut noter que le terme qui suit le terme de rang 3 millions de la suite Fibonacci est de l'ordre de 10^{626963} ! La ligne 29 du code (Figure 2) permet d'afficher les 10 derniers chiffres après la virgule de Φ jusqu'au millionième chiffre.

Figure 3 et 4 présentent un extrait des 4,500 premiers (respectivement derniers) chiffres de notre estimation de Φ suite à l'exécution de notre code Matlab. Le résultat complet est accessible dans [5].

Il est à noter que le temps de calcul du code, donné dans la Figure 2, augmente de plus en plus tant qu'on avance dans la génération des termes de la suite Fibonacci. Car même si l'opération informatique se restreint à l'addition entre deux termes consécutifs, les deux termes impliqués dans cette opération deviennent de plus en plus très grands. Faire la somme de deux très grands nombres est similaire à effectuer une multitude d'additions de plus petits nombres : cela nécéssite donc du temps et de la mémoire.

4 Évaluation du Temps d'Exécution

La figure 5 représente une évaluation du temps d'exécution de notre simple code Matlab qui adapte l'algorithme I [5]. Dans cette figure, nous avons mesuré le temps d'exécution nécessaire à l'accomplissement de chaque groupe de 100,000 itérations du code Matlab avec un temps total accumulé de 10,47 heures. Rappelons que chaque itération ne comporte que : une addition (a+b) et trois af-

4. Matlab prévoit des fonctionnalités de traitement du big data et des très grands nombres [6], ce qui n'a pas été exploité dans cet article.

5. Machine utilisée : Système - Mac OS X, version 10.9.5 ; Processeur - 2.7 GHz Intel Core i7 ; Mémoire - 8 Go ; Matlab - version 8.3.0.532, R2014a, 64 bit

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimation du nombre d'or.
% Approximation par la suite de Fibonacci U[n+1]/U[n]
% Auteur : Tayeb Lemlouma, Tayeb.Lemlouma@irisa.fr, 02/2015 (c)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

syms a b phi
a=sym('1');
b=sym('1');

fprintf('Staring time: %s\n',datestr(now));
for i=1:3000000
    % Generation des termes de la suite Fibonacci
    c=a+b;
    a=b;
    b=c;

    % Afficher la progression
    if mod(i,100000) == 0
        fprintf('Current progression (step of 100000): %d\n',i);
        fprintf('Current time: %s\n',datestr(now));
    end
end

phi=sym(b/a);
x = vpa(phi, 1000010);
phiStr=char(x);

fprintf('Last 1,000,000 numbers are: %s\n',phiStr(999993:1000002));

fprintf('End time: %s\n',datestr(now));
```

FIGURE 2 – code informatique avec le langage Matlab pour l'estimation de Φ

fectations (c=, a=, b=) "utiles" ! On note que le temps nécessaire au calcul des premiers groupes de 100,000 itérations est beaucoup plus petit que le temps nécessaire pour les derniers groupes. A vrai dire, même en apparence (c'est à dire sans rentrer dans le détail de comment Matlab gère ses calculs et sa mémoire), le code nécessaire au calcul final et au stockage mémoire progressif des temps de chaque groupe a nécessité plus de 4 opérations informatiques. La figure 6 donne le code utilisé pour obtenir les résultats présentés dans la figure 5. Notons donc, qu'il y a, entre autre, 3 millions d'opérations relatives à : l'incrémentation de la variable i, la division de i par 100,000, la vérification du reste de la division (nul ou pas), les affichages, l'extraction du temps courant (lecture, soustraction, affectations, incrémentation) et le stockage du temps de calcul pour chaque groupe.

La figure 8 représente une évaluation du temps d'exécution avec un autre code que nous proposons en adaptant notre algorithme I [6]. Ce code, donné par la figure 7, calcule 30 millions de termes de la suite Fibonacci au lieu de 3 millions calculés avec notre précédent code Matlab (figure 5). Incontestablement, ce code

6. Code avec Wolfram Mathematica sous les mêmes conditions matérielles que le précédent code

FIGURE 3 – le résultat des 4,500 premiers chiffres extraits du million de chiffres après la virgule de notre estimation de Φ

	1	**11**	**21**	**31**	**41**	**51**	**61**	**71**	**81**	**91**
0	6180339887	4989484820	4586834365	6381177203	0917980576	2862135448	6227052604	6281890244	9707207204	1893911374
100	8475408807	5386891752	1266338622	2353693179	3180060766	7263544333	8908659593	9582905638	3226613199	2829026788
200	0675208766	8925017116	9620703222	1043216269	5486262963	1361443814	9758701220	3408058879	5445474924	6185695364
300	8644492410	4432077134	4947049565	8467885098	7433944221	2544877066	4780915884	6074998871	2400765217	0575179788
400	3416625624	9407589069	7040002812	1042762177	1117778053	1531714101	1704666599	1466979873	1761356006	7087480710
500	1317952368	9427521948	4353056783	0022878569	9782977834	7845878228	9110976250	0302696156	1700250464	3382437764
600	8610283831	2683303724	2926752631	1653392473	1671112115	8818638513	3162038400	5222165791	2866752946	5490681131
700	7159934323	5973494985	0904094762	1322298101	7261070596	1164562990	9816290555	2085247903	5240602017	2799747175
800	3427775927	7862561943	2082750513	1218156285	5122248093	9471234145	1702237358	0577278616	0086883829	5230459264
900	7878017889	9219902707	7690389532	1968198615	1437803149	9741106926	0886742962	2675756052	3172777520	3536139362
1000	1076738937	6455606060	5921658946	6759551900	4005559089	5022953094	2312482355	2122124154	4400647034	0565734797
1100	6639723949	4994658457	8873039623	0903750339	9385621024	2369025138	6804145779	9569812244	5747178034	1731264532
1200	2041639723	2134044449	4873023154	1767689375	2103068737	8803441700	9395440962	7955898678	7232095124	2689355730
1300	9704509595	6844017555	1988192180	2064052905	5189349475	9260073485	2282101088	1946445442	2231889131	9294689622
1400	0023014437	7026992300	7803085261	1807545192	8877050210	9684249362	7135925187	6077788466	5836150238	9134933331
1500	2231053392	3213624319	2637289106	7050339928	2265263556	2090297986	4247275977	2565508615	4875435748	2647181414
1600	5127000602	3890162077	7322449943	5308899909	5016803281	1219432048	1964387675	8633147985	7191139781	5397807476
1700	1507722117	5082694586	3932045652	0989698555	6781410696	8372884058	7461033781	0544439094	3683583581	3811311689
1800	9385557697	5484149144	5341509129	5407005019	4775486163	0754226417	2939468036	7319805861	8339183285	9913039607
1900	2014455950	4497792120	7612478564	5916160837	0594987860	0697018940	9886400764	4361709334	1727091914	3365013715
2000	7660114803	8143062623	8051432117	3481510055	9013456101	1800790506	3814215270	9308588092	8757034505	0780814545
2100	8819906336	1298279814	1174533927	3120809289	7279222132	9806429468	7824274874	0174505540	6778757083	2373109759
2200	1511776297	8443284747	9081765180	9778726841	6117632503	8612112914	3683437670	2350371116	3307258698	8325871033
2300	6322238109	8090121101	9899176841	4917512331	3401527338	4383723450	0934786049	7929459915	8220125810	4598230925
2400	5287212413	7043614910	2054718554	9611808764	2657651106	0545881475	6044317847	9858453973	1286301625	4487611485
2500	2021706440	4111660766	9505977578	3257039511	0878230827	1064789390	2111569103	9276838453	8633332156	5829659773
2600	1034360323	2254574363	7204124406	4088826737	5843395367	9593123221	3437320995	7498894699	5656473600	7295999839
2700	1288103197	4263125179	7141432012	3112795518	9477817269	1415891177	9919564812	5580018455	0656329528	5985910009
2800	0862180297	7563789259	9916499464	2819302229	3552346674	7593269516	5421402109	1363018194	7227078901	2208728736
2900	1707348649	9981562554	7281137347	9871656952	7489008144	3840532748	3781378246	6917444229	6349147081	5700735254
3000	5707089772	6754693438	2261954686	1533120953	3579238014	6092735102	1011919021	8360675097	3089575289	5774681422
3100	9543394385	4931553396	3038072916	9175846101	4609950550	6480367930	4147236572	0398600735	5076090231	7312501613
3200	2048435836	4817704848	1810991602	4425232716	7219018933	4596378608	7875287017	3935930301	3359011237	1023917126
3300	5904702634	9402830766	8767436386	5132710628	0323174069	3173344823	4356453185	0581353108	5497333507	5996677871
3400	2449058363	6754132890	8624063245	6395357212	5242611702	7802865604	3234942837	3017255744	0583727826	7996031739
3500	3640132876	2770124367	9831144643	6947670531	2724924104	7167001382	4783128656	5064934341	8039004101	7805339505
3600	8772458665	5755229391	5823970841	7729833728	2311525692	6092995942	2400005606	2667867435	7923972454	0848176519
3700	7343626526	8944888552	7202747787	4733598353	6727761407	5917120513	2693448375	2991649980	9360246178	4426757277
3800	6790019191	9070380522	0461232482	3913261043	2719168451	2306023627	8935454324	6176997575	3689041763	6502547851
3900	3824631465	8336383376	0235778992	6729886321	6185839590	3639981838	4582764491	2459809370	4305555961	3797343261
4000	3483049494	9686810895	3569634828	1781288625	3646084203	3946538194	4194571426	6682371839	4918323709	0857485026
4100	6568039897	4406621053	6030640026	0817112665	9954199368	7316094572	2888109207	7882277203	6366844815	3256172841
4200	1769097926	6665522384	6883113718	5299192163	1905201568	6312228207	1559987646	8423552059	2853717578	0765605036
4300	7731309751	9122397388	7224682580	5715974457	4048429878	0735221598	4266766257	8077062019	4304005425	5015831250
4400	3017534094	1171910192	9890384472	5033298802	4501436796	8441694795	9545304591	0313811621	8704567997	8663661746
4500	0595700034	4597011352	5181346006	5655352034	7888117414	9941274826	4152135567	7639403907	1038708818	2338068033

FIGURE 4 – le résultat des 4,500 derniers chiffres extraits du million de chiffres après la virgule de notre estimation de Φ

	1	11	21	31	41	51	61	71	81	91
995500	9220901185	2648770809	0671755561	3735815385	6578653249	1246846535	0841130995	7947476734	8691626685	3041342706
995600	1488060358	1863207085	3214576425	7659430026	4461347299	8915571582	5088478266	2158668433	4539751798	2479995666
995700	7086725759	5561258114	9305765687	1301996388	2800805025	0821461311	2160409589	7112456452	4384766679	4812736512
995800	0230934246	3390671135	6515380707	3800833346	2155699361	5904426801	0711947305	8788447041	6583323808	4504483055
995900	9914760437	2215129819	0162782163	1123369710	9887537652	3470777041	6034080908	0799828579	5319675189	0413019959
996000	1840575332	2503427219	8448966171	5972354148	1400163145	4013883727	6841340361	8779768780	5697405418	2708924009
996100	7150183607	6226318413	9015838875	7514327244	9121011058	7237939531	6739445543	1446234275	0236378492	2889693305
996200	9021581159	3540474827	5785525484	4395828116	8073442662	2587327682	5464183561	2452930797	1468122117	6267928923
996300	6243073171	4743003272	9905119931	0552834601	4526171218	3463679574	4855022548	6014605489	0044028997	1663028113
996400	3578311706	1990023036	4141991588	9393454540	3375059410	0498763683	2218810786	1623118203	8506817973	8583994211
996500	0817432803	4772567845	7243716207	2745674426	3777071046	3124852970	9088702849	1752872353	2118438743	6651613939
996600	5748298139	8318481041	1080294053	2950808767	6145047952	2779068601	5071036529	1101646221	1018426107	4085893371
996700	3650241834	6005864998	6625148603	6732822036	8365271306	0759821519	5902834638	5131721663	9192140760	5035898530
996800	7584266776	3748965239	9182043160	2681388290	9163329738	5352117103	7140375193	9653240474	1437286050	5303861363
996900	3264696060	2612455238	7485727995	2180560287	9607732817	6410294915	3787566379	5434846775	0428960590	0383028636
997000	1936972989	6127798162	9673680544	0391702681	9617024838	2369260006	4276225249	3840915264	8115563647	6486775970
997100	1945655192	4062815881	4516374887	7436845361	2680541779	8766751959	3649611871	2902272255	9648986286	1960130151
997200	9637105885	7155479856	7667105695	1385940623	3591996616	3751981348	8651915999	4007136422	2092910122	6971741276
997300	0695344902	7586815610	8539069637	7343658114	6806252846	7281387606	6200837637	2666284235	0388131581	9201795083
997400	0766028601	9938445495	8304628308	5316478458	7413142662	4155581330	7458083759	1320890572	6649391519	7912363786
997500	9207009578	4962552126	4493788346	7935610655	8055226933	8454409833	9540754709	2648195381	9656183526	8676934247
997600	7205066613	5409341026	0675926374	5105371873	3259627569	8686337234	2157141191	7483178446	4683153103	4655773027
997700	2118892817	8338584762	2237397938	2387595212	8313796656	3400883542	1096048382	3577857622	5051830223	2844400415
997800	2185724529	2842000377	9089872235	3366488861	6796169196	1771137471	4497684930	2510835420	6821010159	6198263838
997900	5820054585	3846313212	1240820681	3991394013	8676515851	9962189950	2467693762	1503784511	9077997541	8513162242
998000	9405353971	0172876199	8301435941	2870775210	3497857214	9267021223	5322160483	2735862381	6270149111	0956690761
998100	5952249194	2939581659	4864635907	4614266725	2960888566	8791580112	9779452814	6468413134	2498256938	1036114366
998200	4182169660	6947623096	9104354244	1317483293	8619848122	0426691946	5596823347	3837777419	4403002151	2505621610
998300	0320383577	8943800671	9585239526	0368357604	3330011159	9422330459	3099265154	8388858011	9045151762	1260377680
998400	4430674653	8647568926	1936909712	1255922371	6069346233	2560068010	0524088922	1982359971	3424290450	4662201296
998500	4901160165	1657424150	5150929654	0885671311	9707254485	5352526903	4453289375	0806441929	0851072524	3842501084
998600	2371010123	8558778805	0690286924	9721057893	2068240814	6764204813	1562759257	1431923440	0308664238	3147583920
998700	1590761904	2553219473	4346444529	1185059989	0847689846	2346443450	3756891371	3746614658	1944464085	7213541188
998800	0421217431	0505615597	1107987443	9338246027	9732579647	4194352831	2529597650	7371760128	2105436966	5124548831
998900	6493140916	9454789489	4812312903	6547683529	5060884606	6123097844	2295187996	7294061891	1706526648	9672355101
999000	1916177829	9482572780	8065379884	7163908322	4603418870	0056842728	7857412034	5844829877	6479993381	9395840272
999100	0393053769	3091980888	8773193784	6481581383	4778669763	0940911671	5601561946	1923513246	1948151475	6926378256
999200	4913910089	3448826972	9626293655	2161805176	3502153817	4612715412	6935600094	4676976293	1211846708	2765982767
999300	5156055778	7616992841	7745105120	5498805479	5149672891	5430299676	9121988102	2631946598	0638046610	7909313528
999400	5795767486	2371564191	2894020926	2506392028	4548765631	6702223544	6762108166	9034116747	3368965489	1698040997
999500	0882257568	4642280648	5457083181	5481725027	3792269673	6727849127	1556586364	0224840546	0631688079	3980641910
999600	0112929897	7864263472	2933479751	5009919519	9175667270	2399802377	9501187453	1421414112	1492356977	9306284560
999700	3141116573	0530575932	7642940870	9647927149	7397271316	9819730050	1906408777	6020193233	3501404222	0648552492
999800	4036963502	0910998952	5877489328	9262893136	0690716416	7236345365	9300940411	5272727958	0441683058	5894338828
999900	3563761751	0166237834	2481029135	0160688497	8280735664	7594202182	9737743512	5197077601	3387228787	4153226344

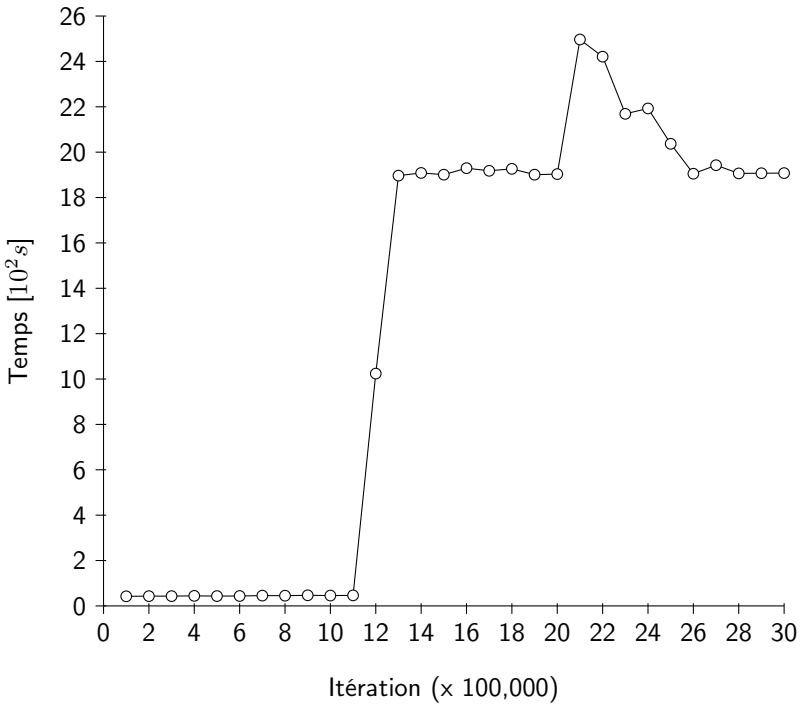

FIGURE 5 – évaluation du temps d'exécution de notre estimation de Φ (3,000,000 termes, total : 10,47 heures)

```
start_time = clock();
TimeVector = zeros(30,1); rankTime=1;
for i=1:3000000
    % Generation des termes de la suite Fibonacci
    c=a+b;
    a=b;
    b=c;

    % Afficher la progression
    if mod(i,100000) ==0
        fprintf('Current progression (step of 100000): %d\n',i);
        elapsedTime = etime(clock(), start_time);
        fprintf('Current time: %s [group dealy: %.2f s]\n',
        datestr(now), elapsedTime);
        start_time = clock();
        TimeVector(rankTime,1)=elapsedTime;
        rankTime=rankTime+1;
    end
end
```

FIGURE 6 – mesure du temps de l'estimation de Φ avec un code Matlab

```
a = 1;
b = 1;
c = 0;
x1 = AbsoluteTime[];
et = Timing[
   Do[
    c = a + b;
    a = b;
    b = c;
    If[Mod[i, 500000] == 0,
     x2 = AbsoluteTime[];
     Print["Iteration: ", i, " Temps : ", (x2 - x1), " secondes "];
     x1 = AbsoluteTime[];
     ],
    {i, 30000000}
    ]
   ];
Part[et, 1]  secondes
```

FIGURE 7 – code avec Mathematica pour l'estimation de Φ

optimise drastiquement le temps d'exécution par rapport au code précédent : 2,32 heures pour 30 millions de termes Fibonacci contre 10,47 heures pour 3 millions de termes !

5 Distribution du Calcul

Si des additions de très grands nombres prennent autant de temps que cela, pouvons nous améliorer ce calcul ? Et si on exécutait une addition de deux grands nombres a et b mais en parallèle ? Pouvons nous espérer une optimisation du

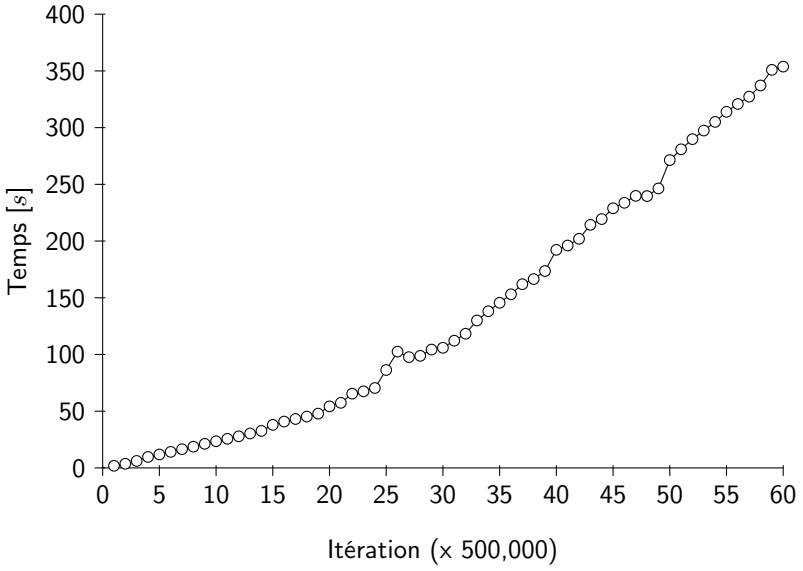

FIGURE 8 – estimation de Φ avec Wolfram Mathematica - (30,000,000 termes, total : 2,32 heures)

temps ? Allons-y avec une méthode très simpliste ! Décomposons la somme de a et b en une multitude d'opérations simples (mais en parallèle) des chiffres de a et de b mais chiffre par chiffre [7] ! Ce qui veut dire qu'au lieu de faire la somme de 1977 et 907 d'un seul coup, on va effectuer l'opération chiffre par chiffre comme suit : 7+7, 7+0, 9+9, 1+0, on fera attention à la retenue qu'on additionnera en parallèle aussi. 1977 + 907 va donc donner un résultat "temporaire" sans retenue de 1874 et une retenue de 1010, la somme de 1874 et 1010 donne bien un résultat correct de 2884. Notons que nous ne sommes pas obligés d'attendre la fin du calcul de la retenue pour entamer le calcul du résultat final ce qui va nous ajouter encore un peu de parallélisme !

La figure 9 présente notre décomposition de l'addition en un ensemble de processus parallèles indépendants [8]. Les cases grises, qui contiendront le résultat final sont initialisées à zéro. Chaque processus indépendant, effectue le même traitement : il additionne deux chiffres et ajoute le résultat sans retenue à la case grise correspondante et la retenue à la case grise voisine. Le processus doit se répéter si l'accumulation du résultat et de la retenue donne une nouvelle retenue.

La figure 10 représente un extrait de code Matlab pour une traduction intuitive de notre méthode. Le code proposé utilise des boucles parallèles (*parfor*). Malheureusement, Matlab implémente, d'une manière limitée, les boucles parallèles et pose des restrictions sur la mémoire partagée [7] [9]. Dans notre cas, la case

7. si on veut généraliser : bloc par bloc et avec plusieurs nombres : $a + b + c + ...$

8. qui peuvent être lancés sur plusieurs processeurs différents locaux ou interconnectés par un réseau

9. Ce code donne l'erreur 'Error : The variable RESULT1 in a *parfor* cannot be classified.'

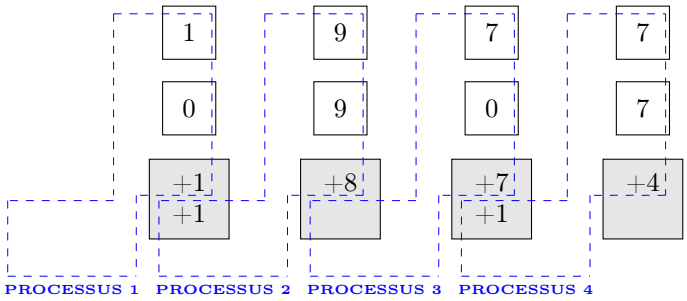

FIGURE 9 – proposition de calcul parallèl de l'addition de très grands nombres

```
1 RESULT1 = zeros(1,maxDigits+1);
2 parfor i = 1:maxDigits
3     %la somme de A (a_Vector(i)) et B (b_Vector(i))
4     t=sprintf('%02.0f',a_Vector(i)+b_Vector(i));
5     RESULT1(i+1) = str2num(t(2));   %le resultat
6     RESULT1(i) = RESULT1(i) + str2num(t(1)); %la retenue
7 end
```

FIGURE 10 – extrait de code pour l'addition de très grands nombres

grise qui accumule la retenue et le résultat de l'addition ne peut donc pas être « touchée » par deux processus parallèles différents (figure 9) ! Dans cet article, et pour donner tant de même une idée d'un code Matlab correct et qui adapte notre méthode de décomposition, nous serons obligés de définir un vecteur de retenue (REMAINING1) qui doit être additionné avec le résultat (RESULT1) (figure 11), en suivant le même prinicipe dans la suite du code. Evidemment, ceci handicape le parallélisme du code et augmente le temps de son exécution.

```
1 RESULT1 = zeros(1,maxDigits+1);
2 REMAINING1 = zeros(1,maxDigits+1);
3 parfor i = 1:maxDigits
4     %la somme de A (a_Vector(i)) et B (b_Vector(i))
5     t=sprintf('%02.0f',a_Vector(i)+b_Vector(i));
6     RESULT1(i+1) = str2num(t(2));   %le resultat
7     REMAINING1 (i) = str2num(t(1)); %la retenue
8 end
```

FIGURE 11 – extrait de code (adapté aux restrictions Matlab) pour l'addition de très grands nombres

à cause de l'implémentation limitée de Matlab des boucles parallèles

6 Calcul Progressif

Lorsqu'on manipule de grosses quantités d'informations ou de très grands nombres nous avons souvent besoin d'obtenir le résultat final d'une manière progressive. C'est à dire, qu'au lieu d'attendre la fin du traitement complet, une application donnée peut avoir besoin d'un résultat partiel qui se complètera au fur et à mesure. Selon les applications, cela peut être fait pour différentes raisons : entamer un traitement partiel de parties indépendantes d'information, effectuer des estimations, faire des sauvegardes, échanger des résultats avec d'autres machines participatives dans le calcul... Afin d'arriver à cet objectif, il est souvent question de repenser les algorithmes intuitifs autrement : il faut donc les adapter pour qu'il fournissent un résultat partiel (mais correct) qui se complète avec le temps, au lieu d'attendre longtemps avant que le résultat final ne tombe d'un coup !

Prenons notre exemple d'approximation de Φ (Algorithme I). Nous avons remarqué que le calcul se base sur la division des deux derniers termes de la suite Fibonacci, ainsi qu'un terme de cette suite se calcule progressivement sur la base des deux derniers termes calculés. Peut on, par exemple, obtenir une partie du résultat final sans pour autant finir le calcul dans sa totalité ? En d'autres termes, peut-on calculer une partie des deux derniers termes de la suite Fibonacci : termes de rangs « 3 millions » et « 3 millions moins un » sans générer la totalité des autres termes précédents (soit 3 millions moins 3 termes) ? La réponse est : oui si on repense notre algorithme autrement.

Considérons les 22 premiers termes des la suite Fibonacci : 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21 – ... – 10946 – 17711. Notre objectif est d'obtenir les termes 10946 et 17711 progressivement, c'est à dire chiffre par chiffre. Par exemple, pour le dernier terme on devrait arriver à obtenir 1 puis 1 puis 7 puis 7 et enfin 1 sans pour autant calculer la totalité des 20 derniers termes de la suite Fibonacci. L'idée proposée ici est de considérer les chiffres qui forment les différents termes colonne par colonne (voir Figure 12). Afin d'obtenir le premier chiffre des deux derniers termes (le 1 du 17711 et le 6 du 10946), nous allons donc faire la génération des termes colonne par colonne en commençant par la colonne la plus à droite. A chaque fin de calcul d'une colonne, nous obtiendrons le chiffre le plus à droite des deux derniers termes : le 6 pour le 10946 et le 1 pour le 17711, puis, le 4 et le 1, puis, le 9 et le 7, ainsi de suite. Notons que cet algorithme ne nécessite que la connaissance de la première colonne et ne requiert pas la mémorisation de la totalité des colonnes. En effet, chaque calcul de colonne va générer seulement une colonne de retenue qui servira au calcul de la colonne suivante. La figure 12 représente le calcul de la première colonne (c'est à dire : 1 – 1 – 2 – 3 – 5 – 8 – 3 – 1 – 4 – 5 – 9 – 4 – 3 – 7 – 0 – 7 – 7 – 4 – 1 – 5 – 6 – 1) jusqu'au rang voulu : le rang 22. Ce nouvel algorithme (Figure 12) :

- mémorisera le résultat partiel des deux derniers termes (ici : 6 et 1)
- identifiera qu'il faut poursuivre le calcul des colonne suivantes car il y a une retenue identifiée à partir du septième terme (rang 7)
- générera une colonne de retenues : 1 – 1 – 0 – 0 – 0 – 1 – 1 – 0 – 1 – 0 – 0 – 1 – 1 – 0 – 0 – 1

Avec chaque colonne de retenues, le calcul se poursuit de la même manière : on mémorise notre résultat partiel (4 et 1 pour la deuxième colonne), on identifie à

partir de quel rang il y a une retenue (rang 12 pour la deuxième colonne), et on génère la colonne de retenues suivantes (exemple, la deuxième colonne génèrera la colonne de retenues : $1-1-0-1-0-0-1-1-1-1-1$). Notons que chaque colonne précédente n'est pas mémorisée et le calcul s'effectue toujours sur la colonne de retenue courante. La figure 13 représente une implémentation Matlab de cette méthode de génération progressive du résultat.

7 Conclusion

Le big data va continuer à présenter des défis aux scientifiques. Son défi majeur étant que le traitement (avec beaucoup de calcul impliquant souvent de très grands nombres) doit se faire en un temps très limité. Sans cette contrainte, un big data serait l'équivalent d'un flux classique de données, qui finissent certes par devenir massives, mais qui peuvent attendre leur tour pour être traitées ! Jour après jour, nous aurons plus de mémoire et plus de capacités de calcul dans nos machines. Hélas, cela ne suffira pas à absorber le temps nécessaire à traiter ces mégadonnées si l'on ne revoit pas nos actuels algorithmes et méthodes de calcul.

Les progrès technologiques ont engendré une évolution de la société en terme de consommation et de production de l'information ou plus généralement de ressources. Tout cela continue à engendrer un « big » big data ! Cette évolution technologique a permis également à notre esprit humain de vérifier et de chercher les secrets de phénomènes inaccessibles auparavant, tels que l'infiniment petit et l'infiniment grand, ce qui implique encore plus de traitement intensif. Enfin, la technologie de nos jours permet aussi de faire parler les machines entre elles (« machine-to-machine ») et de permettre aux plus petits objets et capteurs, qui nous entourent, de participer à cette course à la génération du big data. Ce big data qui ne demande qu'une chose : être « traité » dans un temps imparti malgré sa différence !

Remerciements

Je tiens à remercier Guylaine LOCATELLI, Professeur de Littérature, pour sa disponibilité et sa gentillesse pour la relecture attentive de cet article.

Références

[1] CERN, *Le centre de données du CERN franchit les 100 pétaoctets*, `http://home.web.cern.ch/fr/about/updates/2013/02/cern-data-centre-passes-100-petabytes`, 14 février 2013.

[2] Pritesh Desai, *The Myth of Golden Ratio*, `http://inventikasolutions.com/the-myth-of-golden-ratio`, 2 Septembre 2012.

[3] Haider Mshali, Tayeb Lemlouma, Damien Magoni, *Analysis of Dependency Evaluation Models for eHealth Services*, GLOBECOM'14 - IEEE Global Communications Conference, Austin, TX, USA, December 8-12, 2014.

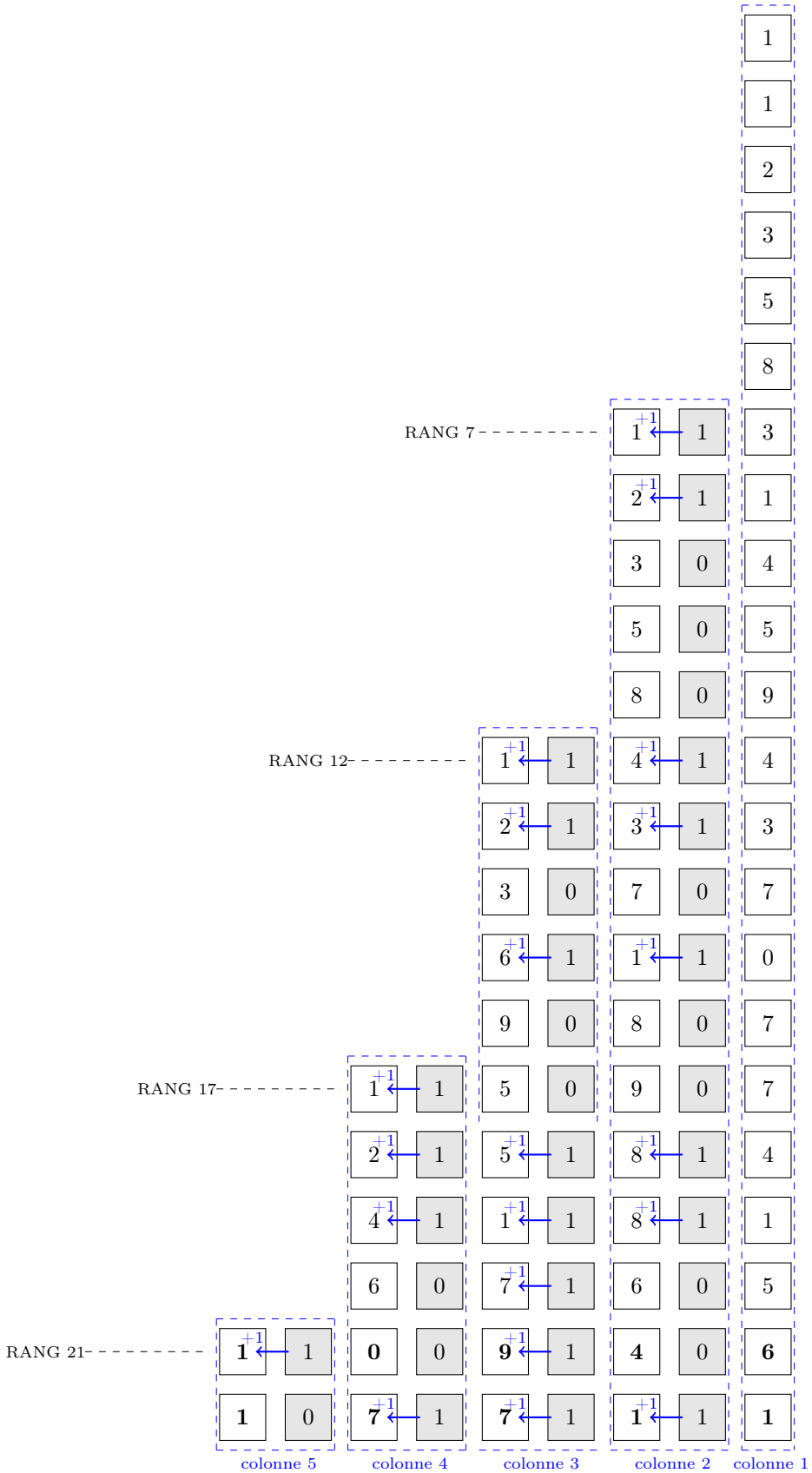

FIGURE 12 – calcul progressif des derniers termes de la suite Fibonacci

[4] Gianluca Fadda, Gian Luca Marcialis, Fabio Roli, Luca Ghiani, *Exploiting the Golden Ratio on Human Faces for Head-Pose Estimation*, Lecture Notes in Computer Science Volume 8156, 2013.

[5] Tayeb Lemlouma, *Estimation du Nombre* Φ *à un Million de Chiffres Après la Virgule*, `http://www.lemlouma.com/papers/EstimationOfPHIandBigData.html`, février 2015.

[6] MathWorks, Big Data et MATLAB, `http://fr.mathworks.com/discovery/big-data-matlab.html`

[7] Dmitry Laptev, *Matlab parallel programming : parfor issues*, `http://blog.dlaptev.org/2014/02/matlab-parallel-programming-parfor.html`

```
ASKED_RANK = 3000000; % nombre de termes voulu
firstTime = 0;
syms nextRowValuesMem nextRowValuesMem2 aValueMem bValueMem
nextRowValuesMem=[]; nextRowValuesMem2=[];
aValueMem=[]; bValueMem=[];
%Generation de la premiere colonne
a=1; b=1;
nextRowRankValue = 0; % a partir de quel rang il y a une retenue dans la prochaine colonne de retenues
i=1;
positionnextRowValuesMem=1; positionnextRowValuesMem2=1;
while i<= ASKED_RANK
    c=a+b;
    if c>=10
        if firstTime == 0
            firstTime =1; % identification du rang ou il y a une retenue dans la prochaine colonne
            nextRowRankValue = i; % on sauvegarde ce rang
        end
        nextRowValuesMem(positionnextRowValuesMem)=1; % on sauvegarde 1 dans la prochaine colonne
        positionnextRowValuesMem=positionnextRowValuesMem+1;
    else
        if firstTime ==1
            nextRowValuesMem(positionnextRowValuesMem)=0; % on sauvegarde 0 dans la prochaine colonne
            positionnextRowValuesMem=positionnextRowValuesMem+1;
        end
    end
    c= mod(c,10); % on ignore le chiffre des decimales dans chaque colonne
    a=b;
    b=c;
    i=i+1;
end
aValueMem=[mod(a,10) aValueMem]; % resultat progressif du dernier terme
bValueMem=[c bValueMem]; % resultat progressif de l'avant dernier terme
row =1;
while nextRowRankValue >0 % tan qu'il y a une retenue, il y a une prochaine colonne a traiter
    i=nextRowRankValue;
    nextRowRankValue=0;
    positionnextRowValuesMem=1;
    positionnextRowValuesMem2=1;
    a=0; b=0;
    firstTime = 0;
    while i<= ASKED_RANK
        VV = nextRowValuesMem(positionnextRowValuesMem);
        positionnextRowValuesMem=positionnextRowValuesMem+1;
            c=a+b+VV;
            if c>=10
                if firstTime == 0
                    firstTime =1;
                    nextRowRankValue = i;
                end
                nextRowValuesMem2(positionnextRowValuesMem2)=1;
                positionnextRowValuesMem2=positionnextRowValuesMem2+1;
            else
                if firstTime ==1
                    nextRowValuesMem2(positionnextRowValuesMem2)=0;
                    positionnextRowValuesMem2=positionnextRowValuesMem2+1;
                end
            end
            c= mod(c,10);
            a=b;
            b=c;
        i=i+1;
    end
    positionnextRowValuesMem=1;
    positionnextRowValuesMem2=1;
    nextRowValuesMem = nextRowValuesMem2; % la nouvelle colonne
    nextRowValuesMem2=[];
    aValueMem=[mod(a,10) aValueMem]; % resultat progressif du dernier terme
    bValueMem=[c bValueMem]; % resultat progressif de l'avant dernier terme
    row=row+1;
end
```

FIGURE 13 – code Matlab pour le calcul progressif des derniers termes de la suite Fibonacci